BRUNO GUILHERME PACCI EVARISTO
LUANA IANARA RUBINI RUIZ

Luva eletrônica inteligente para pacientes com neuropatias periféricas

São Paulo
2016

BRUNO GUILHERME PACCI EVARISTO
LUANA IANARA RUBINI RUIZ

Luva eletrônica inteligente para pacientes com neuropatias periféricas

Trabalho de Conclusão de Curso apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do título de Bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. Francisco Javier Ramirez-Fernandez
Co-orientador: Dr. Wesley Becari

.

São Paulo
2016

BRUNO GUILHERME PACCI EVARISTO
LUANA IANARA RUBINI RUIZ

Luva eletrônica inteligente para pacientes com neuropatias periféricas

Trabalho de Conclusão de Curso apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do título de Bacharel em Engenharia Elétrica.

Área de Concentração: Sistemas Eletrônicos

Orientador: Prof. Dr. Francisco Javier Ramirez-Fernandez
Co-orientador: Dr. Wesley Becari

São Paulo
2016

AGRADECIMENTOS

Ao nosso orientador, Prof. Dr. Francisco Javier Ramirez-Fernandez, e ao nosso co-orientador, Dr. Wesley Becari, pelo tempo e trabalho dedicados em nossa orientação, por todo o conhecido transmitido e pelas valiosas sugestões.

Aos professores Dr. Marcelo Knörich Zuffo e Dr. Antônio Carlos Seabra, orientadores da disciplina, pelos ensinamentos em metodologia da pesquisa e em desenvolvimento de produto e por sempre nos terem mantido motivados.

Ao Jair Pereira de Souza e ao Laboratório de Microeletrônica da USP, pela prestatividade e por todo o auxílio na confecção da luva.

A esta Universidade, seu corpo docente, sua direção e administração, por proporcionarem recursos sem os quais a realização deste projeto não teria sido possível.

Às nossas famílias e amigos, pelo apoio incondicional.

RESUMO

Neuropatias periféricas acometem entre 2% e 8% da população adulta e geralmente estão relacionadas a doenças como diabetes, hanseníase e alcoolismo. A perda de sensibilidade decorrente desse tipo de lesão dificulta a realização de atividades diárias importantes como escrever, dirigir e se alimentar. Nesse contexto, evidencia-se a necessidade de uma luva eletrônica capaz de reproduzir a capacidade sensorial humana no que tange à classificação e à diferenciação de objetos. O objetivo deste projeto foi conceber e construir uma luva eletrônica inteligente de baixo custo a partir de sensores de pressão e de um algoritmo de classificação. Dois sensores capacitivos de pressão foram construídos a partir de lâminas de cobre e EVA (Etil Vinil Acetato) e instalados nos dedos polegar e indicador de uma luva de poliamida e elastano. Esses sensores se comunicam com o computador por meio de uma placa contendo um CDC (*Capacitive-To-Digital Converter*) conectada a uma interface I2C-USB. As leituras de capacitância em níveis digitais são realizadas por um programa em Matlab responsável pela comunicação serial e pela interface utilizando um GUI (*Graphical User Interface*). O GUI é responsável pela interação com o usuário e pela predição do objeto manipulado a partir de um algoritmo classificador do tipo SVM *(Support Vector Machine*) gaussiano médio. Foi possível observar que as medições relativas a diferentes objetos formam *cluster*s distinguíveis e um erro de classificação inferior a 2% foi obtido no treinamento do classificador com 14 objetos. Foram realizados testes com um classificador treinado com 5 objetos (pelúcia, caderno, garrafa cheia, meio cheia e vazia) e todas as predições resultaram corretas. Os resultados permitem estabelecer uma base para propor o desenvolvimento de uma malha de atuação que possa se comunicar com o sistema nervoso do paciente e ser controlada pelo sistema de sensoriamento construído neste projeto.

Palavras-chave: classificadores, sensores táteis, neuropatias periféricas

ABSTRACT

Peripheral neuropathies affect around 2% to 8% of the adult population and are usually caused by diseases such as diabetes, hanseniasis and alcoholism. The sensitivity loss resulting from this type of injury makes it much more difficult to cope with daily activities such as writing, driving and eating. Given the context, the need to develop an electronic glove capable of reproducing the human sensorial capacities regarding the classification and the differentiation of objects is detected. The objective of this project was to design and implement an intelligent low-cost glove using pressure sensors and a classification algorithm. Two capacitive pressure sensors were manufactured from thin copper sheets and a layer of EVA (Ethyl Vinyl Acetate), being subsequently attached to both the thumb and index fingers of a polyamide-elastane glove. These sensors communicate with a computer through a board containing a CDC (Capacitive-To-Digital Converter), which is connected to an I2C-USB interface. The obtained capacitance values, represented in the form of digital levels, are conducted by a program developed in Matlab, which is responsible for both the serial communication and the interface using a GUI (Graphical User Interface). The information collected from these readings is then sent to a GUI (Graphical User Interface) implemented in Matlab. The GUI undertakes the interaction with the user as well as the prediction of the manipulated object through a classification algorithm, which runs using an average gaussian SVM (Support Vector Machine). It was possible to observe that the measurements conducted for different objects gather in clusters, which may be distinguished with an error smaller than 2% for a 14-object-training session using the classification algorithm. Tests using five objects (a teddy bear and a notebook as well as a full, a half-full and an empty water bottle) resulted in a flawless prediction routine. The results make it possible to set a foundation to develop a mechanism that communicates with the patient's nervous system and is controlled by the sensor system developed in this work.

Keywords: classifiers, tactile sensors, peripheral neuropathies

SUMÁRIO

1 Introdução

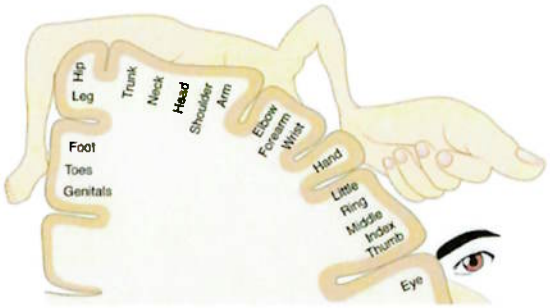
1.1 Identificação do problema e das necessidades

Estima-se que entre 2% e 8% dos adultos sejam acometidos por algum tipo de neuropatia periférica, nome dado às lesões ao sistema nervoso periférico, principal responsável pela capacidade sensorial (ROMAN et al., 2013). Entre crianças e idosos, a incidência dessa patologia é ainda maior.

As causas desse tipo de lesão são variadas. A perda de sensibilidade por neuropatia pode decorrer de trauma direto ou ser um sintoma de doenças sistêmicas ou infecciosas. A hanseníase, maior responsável pela neuropatia periférica no mundo, é um exemplo de doença infecciosa tratável que causa redução ou perda de sensibilidade. Nos países desenvolvidos, o alcoolismo e a diabetes também são grandes causadores dessas lesões. A perda de sensibilidade atinge mais de 60% dos pacientes diabéticos nos Estados Unidos. Ocasionalmente, ela pode se tornar não tratável e levar à amputação (HOSHINO, 2006).

A perda de sensibilidade é um problema especialmente preocupante nas mãos. As pontas dos dedos e as palmas das mãos são, nessa ordem, os membros com maior capacidade sensorial do corpo humano, como se pode ver na Figura 1, que representa um homúnculo - diagrama que exagera as representações dos membros do corpo em função da sua participação no córtex somestésico (SCHOTT, 1993).

Figura 1 - Homúnculo sensorial: representação diagramática proporcional da participação dos membros do corpo no córtex somestésico.



Fonte: Adaptado de http://neurosciencenews.com/files/2014/03/brain-plasticity-in-somatosensory-cortex-after-damage-Homunculus-480x240.jpg (2016).

Evolutivamente, os membros do corpo humano se tornaram mais sensíveis justamente porque são os mais utilizados nas atividades humanas diárias, particularmente as especializadas - o que exige a sensibilidade extremamente fina proporcionada pelos dedos, que chega a 0,15 mm para um ponto isolado (SRINIVASAN et al., 1997). Os problemas decorrentes da sensibilidade reduzida das mãos não se restringem, porém, à incapacidade de reconhecer objetos ou texturas pelo tato, ou de aferir a temperatura de uma superfície. A principal dificuldade decorrente dessas lesões é a incapacidade de segurar objetos de forma estável ou de aplicar a tensão adequada para manipulá-los (SCHMITZ et al., 2010), o que impõe obstáculos consideráveis a tarefas corriqueiras como dirigir, escrever, alimentar-se, entre outras.

Embora a terapia ocupacional seja eficaz na readaptação dos pacientes com neuropatias periféricas, ela apenas permite que se reaprenda a executar determinadas tarefas na condição de lesão, mas não leva à recuperação efetiva da sensibilidade.

1.2 Declaração das necessidades

Nesse contexto, evidencia-se a possibilidade da utilização de luvas eletrônicas acompanhadas ou não de próteses não-invasivas que permitam reproduzir a sensibilidade tátil das mãos. É importante que os sensores utilizados sejam flexíveis ou que possam ser acoplados a materiais flexíveis e que sejam pequenos e ergonômicos o bastante para se adaptarem às pontas dos dedos.

Além disso, há interesse que o sistema não se restrinja a um agrupamento de sensores, mas que se comporte de fato como uma pele eletrônica: da mesma forma que a pele comunica as informações fornecidas pelo tato ao sistema nervoso e que o cérebro aprenda com estas informações, os dados coletados pelos sensores devem ser gerenciados por um *software* capaz de reconhecer e classificar diferentes situações através de métodos de classificação.

1.3 Declaração dos objetivos do projeto

O objetivo deste trabalho é conceber e construir uma luva eletrônica ergonômica e de baixo custo capaz de identificar e diferenciar objetos a partir de sensores de pressão. Por meio da interface adequada, esses sensores se comunicam com um *software* que armazena as informações coletadas e as trata utilizando um algoritmo classificador. O público-alvo do projeto são pessoas cuja capacidade sensorial nas mãos foi afetada por neuropatias periféricas.

2 Estado da arte

2.1 Aplicações e pesquisas

Inúmeros projetos foram desenvolvidos sobre sensores táteis com as mais variadas aplicações. Graças a essa diversidade de aplicações que cada projeto de pele eletrônica é diferente dos demais. A nanotecnologia e o advento da eletrônica em substratos poliméricos (SCHWARTZ et al., 2013) também são responsáveis pelo caráter inovador de vários desses projetos.

Grande parte das peles eletrônicas é projetada para aplicações nas áreas de saúde e medicina. Um exemplo são as palmilhas eletrônicas constituídas por sensores de pressão e emissores de *lasers* em uma faixa de frequências capaz de prevenir a neoformação tecidual nos pés diabéticos (REIS, 2010). Outro são as interfaces ultrafinas com a pele para medição de variáveis fisiológicas e estimulação localizada em substituição a eletrodos (KIM, 2011). No contexto das luvas eletrônicas, que são o escopo desse projeto, estão sendo implementadas luvas com sensores capacitivos para aumentar a sensibilidade de cirurgiões, aferir as propriedades elétricas de tecidos e agrupar em uma única ferramenta funcionalidades de várias que, atualmente, precisam ser manipuladas separadamente (INOVAÇÃO TECNOLÓGICA, 2012). O grande desafio é desenvolver peles macias e integráveis com o silício semicondutor e estender a utilização da pele eletrônica para outros órgãos como, por exemplo, o monitoramento do coração (INOVAÇÃO TECNOLÓGICA, 2012).

No campo das tecnologias interativas, pesquisadores desenvolveram uma pele eletrônica permitindo a visualização instantânea da pressão aplicada por meio de LEDs. Aplicações em potencial são painéis de controle automotivos e interfaces interativas (WANG et al., 2013).

Em robótica, o interesse é replicar com o máximo de fidelidade a pele e o tato humanos. É o caso das mãos do robô humanóide *iCub*, mais especificamente dos

seus dedos. As pontas dos dedos desse robô foram implementadas com placas de circuito impresso flexíveis adaptadas a suportes circulares que imitam a forma das pontas dos dedos humanos. A cada placa está acoplada uma camada de espuma de silicone deformável com propriedades dielétricas, seguida de um composto condutivo de silicone conectado à referência de terra. Trata-se de uma realização de sensores capacitivos de pressão (SCHMITZ et al., 2010).

Cada dedo é formado de 12 *patches* circulares, isto é, 12 capacitores. Esses capacitores, por sua vez, são ligados aos 12 canais de entrada de um CDC, permitindo transmitir as informações coletadas para um computador por via serial. O principal interesse em realizar a conversão o mais próximo possível do transdutor é, nesse caso, a melhoria do nível de sinal-ruído (SCHMITZ et al., 2010).

Tal realização permite ao robô *iCub* aplicar a tensão necessária e suficiente para pegar e sustentar diferentes objetos, sendo equivalente ao elemento sensor de uma malha de controle fechada. Alguns exemplos de objetos utilizados na pesquisa foram ursos de pelúcia, garrafas d'água cheias e vazias, entre outros. O comportamento do robô é análogo ao do ser humano, que avalia a força necessária para segurar um objeto a partir das informações fornecidas pelo tato. Devido ao sistema de sensores capacitivos acoplados a seus dedos o robô é capaz de segurar esses objetos de forma estável, sem deformá-los ou deixá-los escorregar.

2.2 Tecnologias relevantes

2.2.1 Sensores táteis

Sensores táteis são constituídos de arranjos de sensores pontuais de toque, que podem detectar e medir a força de contato em um ponto determinado (CROWDER, 2016). Existem várias realizações desse tipo de sensor. As mais relevantes são listadas abaixo.

Sensores mecânicos: o funcionamento desses sensores de toque é binário e se baseia na ativação de uma microchave mecânica pela força de contato (CROWDER, 2016).

Sensores resistivos: amplamente utilizados no meio industrial por sua simplicidade, esses sensores se baseiam no uso de materiais com uma característica força-resistência bem definida. As forças de contato deformam o material variando os valores de sua resistência elétrica (CROWDER, 2016).

Resistores detectores de força: constituídos de polímeros piezorresistivos condutores, cuja resistência elétrica varia de acordo com a força aplicada em sua superfície. Requerem uma interface simples e operam bem em diferentes ambientes (CROWDER, 2016).

Sensores magnéticos: existem dois tipos de sensores magnéticos. O primeiro deles tem seu funcionamento fundado na variação de fluxo magnético ocasionada pela movimentação de um pequeno ímã mediante a aplicação de uma força. O segundo pode ser um indutor ou transformador cujo núcleo seja fabricado a partir de um material magnetoelástico, deformável sob pressão e, portanto, capaz de provocar mudanças no acoplamento das bobinas (CROWDER, 2016).

Sensores ópticos: existem dois tipos de sensores ópticos. Um deles se baseia na modulação da intensidade da luz pela criação de uma obstrução no canal de propagação, que é construído no interior de um tubo deformável. O outro atua por meio da fotoelasticidade, propriedade de materiais capazes de atenuar a intensidade da luz passante quando deformados. Também existem várias implementações em fibra ótica (CROWDER, 2016).

Sensores piezoelétricos: nos materiais piezoelétricos, a aplicação de uma pressão gera um estresse que por sua vez é convertido em uma tensão elétrica. Esses sensores são normalmente construídos a partir de polímeros com propriedades piezoelétricas, devido à sua maior elasticidade. Um exemplo é o fluoreto de polivinilideno (CROWDER, 2016).

Sensores capacitivos: analogamente aos sensores resistivos, a operação dos sensores capacitivos é baseada na deformação de sua seção transversal ou da distância entre as placas pelas forças de contato. Essa deformação resulta em uma mudança no valor da capacitância. Cada valor de capacitância está associado a uma intensidade bem definida da força de contato. Normalmente, esses sensores têm uma seção transversal extensa para diminuir a influência do ruído e garantir uma boa precisão. Além disso, possuem alta sensibilidade e baixo consumo de energia. É por essa razão que os sensores capacitivos são amplamente utilizados em aplicações biomédicas (PUERS, 1993).

Existem vários tipos de realizações de sensores capacitivos de pressão. O mais comum é o que consiste em uma placa dielétrica envolta por dois eletrodos. Os métodos de fabricação e materiais utilizados são diversos, podendo ir desde a fundição de metal sobre uma camada de vidro ao desenvolvimento de uma fina lâmina condutora sobre vidro fundido (PUERS, 1993).

Outro tipo de sensor capacitivo é aquele cujo funcionamento se baseia no toque. O diafragma e o eletrodo inferior não se tocam na situação de repouso; só há contato entre eles quando uma pressão é aplicada (KO et al., 1999).

2.2.2 Conversores de capacitância

O funcionamento dos conversores de capacitância, analógicos ou digitais, baseia-se na excitação do sensor capacitivo seguida da medição da variação de capacitância por meio de variações de tensão, corrente, frequência ou largura de pulso. As realizações mais comuns desse tipo de conversores são:

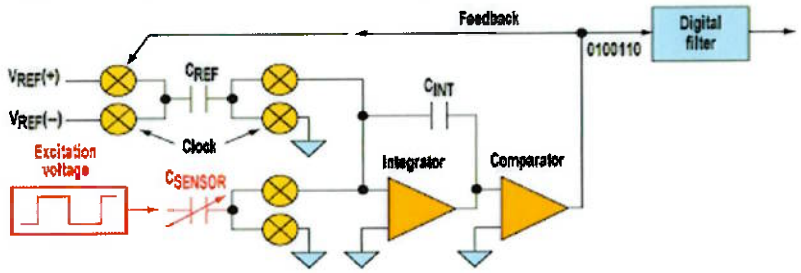
- Excitação do capacitor por uma fonte de corrente por um tempo determinado e posterior medida da tensão no capacitor. Esse método requer uma fonte de corrente de baixa intensidade e alta precisão, e uma alta impedância de entrada para a medida de tensão.

- O capacitor variável é conectado a um resistor para formar um oscilador RC. As variações de capacitância são então medidas por variações na constante de tempo, na frequência ou no período. A desvantagem desse método é sua baixa precisão.
- As variações de capacitância também podem ser obtidas por meio de medições da impedância AC (*Alternating Current*) dos capacitores quando excitados por uma fonte de tensão senoidal. No entanto, esse método requer um circuito eletrônico complexo.
- O método mais utilizado em circuitos integrados, atualmente, é a medida da tensão na saída de um sistema amplificador de carga com um capacitor de referência (BRYCHTA, 2005).

Embora a escolha entre sensor analógico e digital dependa da aplicação, a utilização de CDCs é mais interessante por dois motivos: primeiro porque evita o cascateamento de dois CIs (conversor de capacitância analógico e conversor AD (Analógico-Digital)), além disso possibilita a linearização do sensor, a compensação de temperatura e a calibração do sistema são muito mais fáceis de serem realizadas no domínio digital (BRYCHTA, 2005).

A maior parte dos sistemas de conversão AD emprega conversores sigma-delta, que proporcionam alto desempenho em termos de linearidade e precisão (BRYCHTA, 2005). Os CDCs como o AD7147 da Analog Devices são resultado da integração de um amplificador de carga com um modulador sigma-delta, como mostra a Figura 2.

Figura 2 - Realização de um CDC a partir de um amplificador de carga e um modulador sigma-delta.



Fonte: Adaptado de http://electronicdesign.com

Além das vantagens inerentes aos conversores AD realizados com moduladores sigma-delta, esse sistema tem como pontos fortes o fato de não ser sensível à capacitância entre os sensores e a referência do terra ou às correntes de fuga; e o fato de poder ser fabricado em um único circuito integrado (CI), apresentando alta integração, fácil implementação, repetitividade, confiabilidade e baixo custo (BRYCHTA, 2005).

2.2.3 Aprendizagem de máquina e métodos de classificação

Uma solução de aprendizagem de máquina pode ser resumida como uma descrição concisa de um conjunto dados a partir de uma amostra de tamanho limitado desses dados. Se as amostras em questão têm um padrão entrada-saída, essa descrição é na verdade uma função capaz de predizer saídas para diferentes entradas. Nesses casos, a aprendizagem é dita supervisionada porque os objetos em consideração podem ser associados com classes ou valores numéricos (HERBRICH, 2002).

Os problemas de aprendizagem se assemelham a inferências estatísticas. A única diferença entre eles é que a aprendizagem não requer modelos probabilísticos, pois apenas prediz novas instâncias. Essa tarefa demanda muito menos exemplos para atingir um determinado grau de desempenho e, portanto, é muito mais simples (HERBRICH, 2002).

Nos métodos de aprendizagem supervisionada, os dados amostrais (contendo entradas e saídas) também são referidos como o conjunto de treinamento. O problema é então encontrar a função determinística que minimiza as discrepâncias desse conjunto com as observações futuras de entrada-saída. Quando a saída não é exatamente uma variável, mas sim a resposta à pergunta se dois elementos são iguais ou não, o problema é dito de classificação (HERBRICH, 2002). Métodos de classificação são algoritmos capazes de predizer a classe à qual pertence um dado de entrada a partir de um treinamento prévio com uma coleção de dados pertencentes a classes bem definidas e descritos pelos valores de um conjunto fixo de atributos (WU et al., 2008).

Os principais métodos de classificação são: *k-Nearest Neighbor, Naïve Bayes, Classification Trees, Ensemble e Support Vector Machine (SVM).* Não existe um método melhor do que outro em termos absolutos; o que se tem são métodos adaptados para o tipo de dados que se deseja classificar e implementações robustas de cada um desses métodos.

Na teoria de aprendizagem, calcula-se o limite superior da perda esperada da função treinada, pois ele é uma medida direta do desempenho do algoritmo. Dessa forma, o melhor algoritmo para determinada aplicação pode ser definido por meio do cálculo desse parâmetro. Outro papel importante da perda esperada de um algoritmo é determinar, a partir de uma valor fixo de perda definido, o tamanho ideal do conjunto de treinamento (HERBRICH, 2002). Ele não pode ser muito pequeno para não omitir informações sobre as classes, mas nem muito grande, pois pode ocorrer *overfitting*. A seguir são apresentados alguns algoritmos de classificação.

k-Nearest Neighbor: nesse método, a classe à qual pertencerá a amostra de entrada em questão não é definida pelos seus atributos, mas pela classe predominante entre seus k vizinhos mais próximos. Para aplicação desse método, é importante definir k e uma métrica para a distância entre as amostras (por exemplo, distância euclidiana em 2 dimensões). Além disso, um conjunto de amostras previamente atribuídas a suas respectivas classes, isto é, um conjunto de treinamento, se faz necessário (WU et al., 2008).

Naïve Bayes: neste método, supõe-se que cada amostra possui um vetor de variáveis características. De acordo com o valor de algumas dessas variáveis, que são presumidas independentes, as amostras-treino são atribuídas a diferentes classes pela computação de um *score*. A classificação das amostras em teste se dá, então, pela computação de um *score* individual para cada amostra a partir dos valores de suas variáveis (WU et al., 2008).

Classification Trees: as árvores de classificação são construídas a partir de variáveis comuns às amostras; cada nível de uma árvore corresponde a um novo limiar para o valor de uma ou mais variáveis. As folhas da árvore são as classes de equivalência.

Ensemble: utiliza uma combinação de algoritmos de aprendizado para realizar a classificação, de forma a obter resultados melhores do que os obtidos com o uso individual de tais algoritmos (WU et al., 2008).

Support Vector Machine: geometricamente, a separação entre as diferentes classes é feita por um hiperplano passando entre elas. Esse hiperplano possui a propriedade de maximizar a margem entre as duas classes, isto é, ele é tal que a distância dos elementos mais próximos ao hiperplano até ele é máxima. O cálculo do hiperplano ótimo é um problema de otimização quadrática.

Os pontos definindo a margem, isto é, localizados em suas bordas, são chamados de vetores-suporte, e o meio da margem é o hiperplano ótimo de separação. Apesar de se tratar de um algoritmo de classificação binária, ele pode ser adaptado para lidar com mais de duas classes de separação (MEYER, 2015). Nesses casos, basta dividir o problema de separação entre múltiplas classes em uma série de problemas de separação binária.

O método SVM se baseia no conceito do *kernel* como uma medida da similaridade entre objetos para avaliar a semelhança das componentes de entrada x das amostras de treinamento (x, y) com as amostras x' a serem classificadas. Assim, será possível escolher y' tal que (x', y') seja o mais próximo das amostras de treinamento (SCHÖLKOPF et al., 2002).

O exemplo mais simples de um *kernel* é o produto escalar, $k(x,x') = < x,x' >$. Nesse caso, é preciso que as amostras de entrada sejam representáveis em um espaço vetorial admitindo produto escalar. Caso elas não sejam, utiliza-se uma função de mapeamento $\Phi(x)$ para esse fim. Outra característica é a função *kernel*, que permite encontrar o hiperplano de separação ótimo sem a necessidade de mapeamento explícito (SCHÖLKOPF et al., 2002). A classe dos hiperplanos em um espaço H admitindo produto escalar pode ser definida pela expressão matemática: $< w, \boldsymbol{x} > + b = 0$, onde $\boldsymbol{x}$ e w pertencem a H e b é um número real ($\boldsymbol{x}$ denota $\boldsymbol{x}$ mapeado no espaço $\boldsymbol{H}$). Isso corresponde às funções de decisão da forma $f(\boldsymbol{x}) = sgn(< \boldsymbol{x}, w > + b)$. O hiperplano ótimo de separação é então encontrado pela resolução para w e b do problema de otimização $max(min\{ ||\boldsymbol{x} - \boldsymbol{xi}|| \mid \boldsymbol{x} \epsilon H, < w, \boldsymbol{x} > + b = 0\})$ (SCHÖLKOPF et al., 2002).

Há uma grande variedade de funções *kernel*. Para citar alguns exemplos, o *kernel* gaussiano de base radial e os polinomiais, entre outras. As vantagens desse método são o fato de ele apresentar uma base teórica bem fundada, adaptar-se bem aos casos gerais (não-lineares, bastando projetar os dados em um espaço de dimensão superior até que um hiperplano linear possa ser encontrado), necessitar de poucas amostras de treinamento e não ter sua performance afetada pelo número de dimensões (WU et al., 2008).

O valor de w que é solução desse problema é o vetor normal ao plano de separação, e geometricamente pode-se observar que a margem do hiperplano é dada por $1 / ||w||$. Assim, o problema de otimização para encontrar o hiperplano ótimo de separação pode ser reescrito como $min(||w||^2 \div 2)$ com relação a w e b e sujeito a $yi . (< w, \boldsymbol{xi} > + b) \geq 1$ (SCHÖLKOPF et al., 2002).

Na prática, o hiperplano ótimo de separação pode não existir. Isso acontece quando existe muita sobreposição entre as classes devido a um nível de ruído elevado, por exemplo. Nessas situações, a restrição $yi . (< w, \boldsymbol{xi} > + b) \geq 1$ pode ser relaxada pela introdução de variáveis de folga ξ_i: $y_i . (< w, \boldsymbol{x_i} > + b) \geq 1 - \xi_i$. Assim, uma

margem mais flexível pode ser obtida pela minimização de $||w||^2 \div 2 + C.\Sigma\xi_i,\ C > 0$ (SCHÖLKOPF et al., 2002).

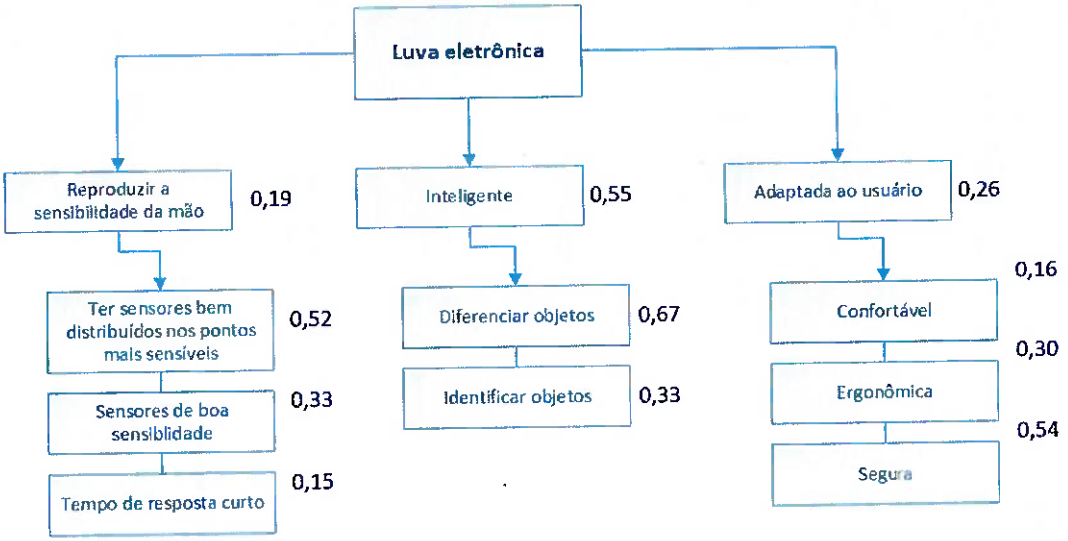
3 Análise dos requisitos de engenharia

Inicialmente, fez-se um levantamento dos objetivos, dos requisitos de engenharia e das suas restrições, e, por último, optou-se por definir a estrutura da eletrônica a ser desenvolvida, bem como do algoritmo de classificação.

3.1 Árvore de objetivos

A Figura 3 apresenta a árvore de objetivos com pesos normalizados. Nela são descritos todos os requisitos e seus pesos, ou seja, a importância relativa dos requisitos propostos.

Figura 3 - Árvore de objetivos com pesos normalizados.



Fonte: Autores.

A Tabela 1 demonstra o cálculo dos pesos descritos na árvore de objetivos. Ressalta-se que o valor "1" descreve igual importância entre os requisitos; o valor "2" indica maior importância; "3" indica muito maior importância; "0,5" indica menor importância; e "0.33" indica muito menor importância.

Tabela 1 - Cálculo dos pesos da árvore de objetivos.

	Reproduzir a sensibilidade da mão	Inteligente	Adaptada ao usuário	Importância relativa	Normalizada
Reproduzir a sensibilidade da mão	1	0,5	0,5	0,6299605249	**0,1898764058**
Inteligente	3	1	2	1,817120593	**0,5476983292**
Adaptada ao usuário	2	0,33	1	0,8706587691	**0,262425265**

Reproduzir a sensibilidade da mão

	Distribuição dos sensores	Sensibilidade dos sensores	Tempo de resposta	Importância relativa	Normalizada
Distribuição dos sensores	1	2	3	1,817120593	**0,5176516463**
Sensibilidade dos sensores	0,5	1	3	1,144714243	**0,3261001029**
Tempo de resposta	0,33	0,5	1	0,5484806552	**0,1562482508**

Inteligente

	Diferenciar objetos	Identificar objetos	Importância relativa	Normalizada
Diferenciar objetos	1	2	1,414213562	**0,6666666667**
Identificar objetos	0,5	1	0,7071067812	**0,3333333333**

Adaptada ao usuário

	Confortável	Ergonômica	Segura	Importância relativa	Normalizada
Confortável	1	0,5	0,33	0,5484806552	**0,1629666187**
Ergonômica	2	1	0,5	1	**0,297123731**
Segura	3	2	1	1,817120593	**0,5399096503**

Fonte: Autores.

3.2 Requisitos de engenharia

Os requisitos de engenharia do presente projeto podem ser divididos em requisitos de usuário, restrições ambientais e requisitos técnicos. Eles estão listados nas Tabelas 2 a 4, respectivamente.

Tabela 2 - Requisitos de usuário e validação.

Requisito	Validação
A luva eletrônica deve ser confortável e ergonômica: tem de se encaixar bem à mão, permitindo que ela se movimente e manipule objetos sem incomodar o usuário. Os cabos ligando a mão ao computador devem ter o tamanho adequado para que o usuário não tenha seus movimentos restringidos.	Uma vez confeccionada a luva, usuários diferentes a experimentarão e avaliarão sua adaptação à mão e se seu uso causa algum desconforto.
O material da luva deve ser inerte e isolante, para que o usuário não desenvolva irritações ou corra risco de choque elétrico.	A resistência de isolamento será medida com o megômetro apropriado. Também serão feitos testes de continuidade DC com um miliômetro para verificar o aterramento do sistema.
O tempo de resposta do sistema de sensores, de transmissão e de classificação deve ser o menos perceptível possível para o usuário, preferencialmente, inferior ao tempo de reação humano médio, de 275 ms (HUMAN BENCHMARK).	As medidas de tempo de resposta serão feitas para cada um dos componentes isoladamente e para o sistema como um todo, com o auxílio de um gerador de funções e um osciloscópio.
A interface de usuário do *software* deve ser clara e compreensível para usuários leigos, sem conhecimentos em eletrônica ou métodos de classificação.	A facilidade de uso do sistema será avaliada mediante testes de utilização por diferentes usuários.

Fonte: Autores.

Tabela 3 - Restrições ambientais e validação.

Requisito	Validação
A temperatura de operação não deve afetar o comportamento de nenhum dos circuitos integrados (para o CDC AD7147, a temperatura de operação deve estar entre -40 e 105 graus Celsius) (ANALOG DEVICES) nem alterar significativamente a constante dielétrica ou as propriedades físicas (forma, espessura do dielétrico na ausência de pressão) dos sensores capacitivos.	Serão feitos ensaios com os mesmos valores de pressão em diferentes temperaturas para aferir a sensibilidade do sistema e, eventualmente, definir os limites de operação.

Fonte: Autores.

Tabela 4 - Requisitos técnicos e validação.

Requisito	Validação
O comprimento dos cabos ligando os sensores da luva ao computador não pode ser excessivo para evitar a atenuação de sinal além de um valor limite.	A atenuação de sinal será medida pela comparação da intensidade dos sinais na saída dos sensores, na saída do conversor e no computador.
A conversão A/D dos sinais provenientes dos sensores deve ser feita o mais próximo possível dos sensores para evitar a degradação da relação sinal-ruído além de um valor limite.	A distância entre os sensores e o elemento conversor será determinada à partir de medidas da relação sinal-ruído para diferentes configurações.
O erro do classificador deve ser menor do que 15% e o conjunto de treinamento não pode ser grande, a fim de evitar *overfitting*.	Serão realizados testes de classificação com diferentes conjuntos de treinamento e os respectivos erros de classificação serão computados.

Fonte: Autores.

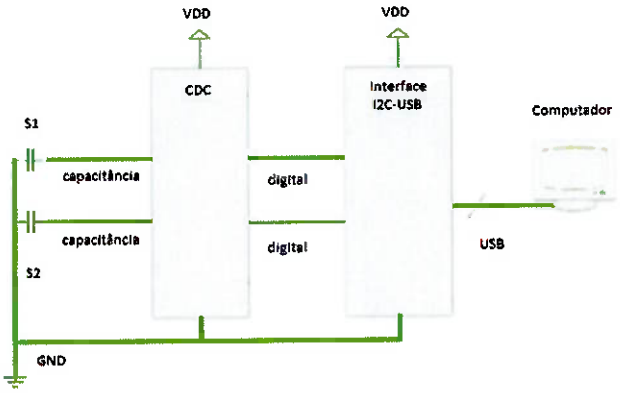
4 Materiais e métodos

Dada a definição dos requisitos, este capítulo apresenta a arquitetura do projeto em uma abordagem começando pela sua decomposição funcional básica e em seguida, partindo para os materiais e métodos empregados. Após a apresentação da descrição funcional e de detalhes do gerenciamento do projeto, serão descritos o desenvolvimento dos sensores, a escolha do elemento de conversão e da interface de comunicação, bem como da implementação do algoritmo de classificação em *software* matemático.

4.1 Decomposição funcional

A Figura 4 apresenta a decomposição funcional do sistema da luva eletrônica. Nela é possível verificar quatro blocos principais: os sensores S1 e S2 desenvolvidos por meio de placas paralelas, o CDC (AD7147, cuja escolha é detalhada nas seções seguintes), a interface I2C-USB, e o computador responsável pela coleta de informações e classificação dos dados coletados. Esses componentes são detalhados na Tabela 5.

Figura 4 - Decomposição funcional da luva eletrônica.



Fonte: Autores.

A luva eletrônica é altamente acoplada, pois tanto os sensores como o CDC são sensíveis às variações de impedâncias capacitivas. Ou seja, todo o sistema sofre influência e interferência das variações de pressão aplicadas em diferentes pontos da luva.

Em relação a coesão, o sistema é moderadamente coeso. Embora os módulos não sejam dedicados, isto é, possam ser utilizados em aplicações variadas, os sensores capacitivos foram desenvolvidos especificamente para este projeto. Além disso, o uso de sensores capacitivos e do CDC reduz as possibilidades de substituição desses módulos sem afetar o sistema. Entre outras palavras, não é possível substituir os sensores capacitivos por piezoelétricos sem substituir o CDC por um conversor AD ou outro circuito de condicionamento.

Tabela 5 - Detalhamento dos blocos da decomposição funcional da luva eletrônica.

Módulo	S (sensores)	CDC: AD7147	Interface I2C-USB	Computador
Entradas	Pressão	Capacitâncias (dos 2 sensores capacitivos instalados na luva)	Entradas digitais I2C	Entrada USB
Saídas	Capacitâncias (dos 2 sensores capacitivos instalados na luva)	Níveis digitais I2C	USB	-
Funcionalidade	Converter valores de pressão em valores de capacitâncias	Converter capacitâncias em tensões elétricas em níveis digitais	Realizar a interface entre linhas I2C e USB para transmitir os dados ao computador	Permitir o armazenamento de dados e a sua classificação pelo uso do Matlab

Fonte: Autores.

4.2 Etapas de desenvolvimento detalhadas

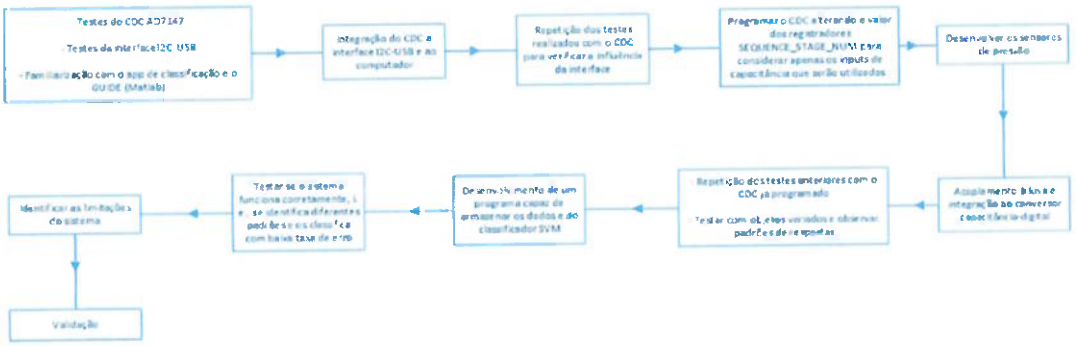
Na Figura 5, são apresentadas as principais atividades relacionadas à construção e à validação do sistema.

A primeira etapa são os testes unitários do CDC AD7147 e da interface I2C-USB e a familiarização com a biblioteca de classificação, o LibSVM e o *GUIDE* (*Graphical User Interface Designer*) do Matlab.

Os testes do AD7147 se baseiam em características específicas desse CI extraídas do seu *datasheet.* Uma informação importante sobre esse CI é a de que ele possui doze entradas de capacitâncias que podem ser lidas em 12 estágios controlados por um sequenciador. Como apresentado na Figura 5, os testes deste componente podem ser subdivididos em:

- Testar os 2 canais de entrada com diferentes valores de capacitâncias, observando as saídas digitais;
- Computar as capacitâncias mínima e máxima detectáveis;
- Observar o tempo de resposta à variação das capacitâncias na entrada;
- Conectar os 2 canais de entrada simultaneamente e observar o comportamento da saída (i.e., do sequenciador);

Figura 5 - Detalhamento da metodologia.



Fonte: Autores.

No que diz respeito à interface I2C-USB, foram instalados no computador os *drivers* necessários e estudar como seria feita a comunicação da porta USB em que a interface foi instalada com o Matlab. A placa de interface foi testada com uma rotina simples (escrita e leitura de um registrador).

No Matlab, o objetivo era realizar a comunicação com as portas USB, estudar o funcionamento dos diversos classificadores, especificamente do SVM, pela análise dos seus respectivos códigos, e desenvolver uma interface gráfica simples utilizando o *GUIDE*, integrando a ela o classificador.

Em seguida, o CDC, a interface e o computador seriam conectados e os testes descritos para o CDC repetidos para calcular a máxima taxa de amostragem, se aplicável, isto é, se limitante face aos requisitos de projeto.

O CDC deveria também ser programado alterando-se o valor dos registradores *SEQUENCE_STAGE_NUM* do AD7147 para considerar apenas as entradas de capacitância que foram utilizadas (2 para os dedos). Se limitante, a taxa de amostragem precisaria ser recalculada.

O passo seguinte era a construção dos sensores capacitivos a partir de placas de metal ou de circuito impresso moldáveis e de algum material dielétrico isotrópico e flexível. Esses sensores têm de ser testados isoladamente e, após integração ao sistema, os testes anteriores são repetidos para computar: as capacitâncias máxima e mínima detectáveis e a resolução dos sensores; o tempo de resposta; e a taxa de amostragem.

Finalmente, o sistema foi testado com objetos variados e os padrões de respostas observados para determinar se os sensores têm saídas diferentes para objetos diferentes.

Isso ocorreu paralelamente ao desenvolvimento do *software* capaz de receber, armazenar e classificar os dados em Matlab. Esse programa possui uma interface gráfica simples, fácil de usar.

A última etapa antes dos testes finais é a confecção da luva em material confortável, isolante e ergonômico. A fim de validar o sistema, foram realizados testes com vários objetos para verificar se a luva é realmente capaz de identificá-los e diferenciá-los com o menor erro possível. Esses testes também têm por objetivo encontrar as limitações do sistema, como objetos irreconhecíveis ou um tempo de reconhecimento acima do ideal, por exemplo.

4.3 Estrutura de custos e matriz de riscos

Os recursos necessários à realização deste projeto são um computador com o *software* Matlab instalado; um CDC AD7147; uma interface USB *all-in-one* (no caso deste projeto, apenas a parte I2C-USB é necessária); e os materiais que foram utilizados na confecção dos sensores.

A Tabela 6 apresenta os custos detalhados de cada um desses componentes.

Tabela 6 - Estrutura de custos.

Componente	**Custo (em dólares)**
Computador + Matlab	Computador pessoal, licença Matlab fornecida pela USP
Interface I2C-USB	13
CDC (5 unidades, em caso de falha)	50
Sensores	7
Total	70

Fonte: Autores

Os riscos mais significativos do projeto são apresentados na Tabela 7, que ilustra a matriz de riscos. Os riscos mais preocupantes são: a presença de defeito nos componentes comprados, que exigiria recompra e atrasaria o projeto, uma vez que esses componentes são importados e têm um tempo de transporte longo; um erro de classificação elevado, que poderia requerer o teste de outros métodos de classificação, o remodelamento dos sensores e/ou o seu reposicionamento na luva.

Também foi identificado um risco de impacto e de probabilidade médios relacionadas à dificuldade em obter resultados semelhantes nas mesmas condições de teste dos sensores capacitivos. A probabilidade de que isso ocorra não é desprezível, pois os sensores capacitivos estão sujeitos à histerese (SCHMITZ et al., 2010).

Tabela 7 - Matriz de riscos.

		Impacto		
		Baixo	Médio	Alto
Probabilidade	Baixo	_____	_____	Erro de classificação elevado
	Médio	_____	Baixa reprodutibilidade e/ou repetitividade nos testes do sensor capacitivo	Componentes comprados apresentam defeito
	Alta	_____	_____	_____

Fonte: Autores.

4.4 Descrição dos blocos constitutivos da luva eletrônica

As opções de desenvolvimento são apresentadas na Tabela 8, e nas seções seguintes são detalhadas as alternativas escolhidas e o seu desenvolvimento.

Tabela 8 - Tabela de conceitos.

Sensores de pressão	**Elemento de conversão**	**Método de classificação**
Capacitivo	Conversor analógico-digital (AD)	*Support Vector Machine*
Piezoelétrico	*Capacitive-to-digital converter (CDC)*	*k-Nearest-Neighbor*
		Árvores de classificação

Fonte: Autores.

4.4.1 Desenvolvimento dos sensores

Entre as opções apresentadas na tabela de conceitos, o tipo de sensor escolhido foi o sensor de pressão capacitivo convencional, por sua simplicidade, alta sensibilidade e baixo consumo de potência. Além disso, essa classe de sensores de pressão é a mais utilizada em aplicações biomédicas.

A escolha desses sensores também é resultado da análise de Pugh, ilustrada nas três tabelas abaixo. A Tabela 9 apresenta os dados mais relevantes de um sensor capacitivo e de um piezoelétrico comerciais.

Tabela 9 - Dados de um sensor capacitivo e de um piezoelétrico.

	Capacitivo	**Piezoelétrico**
Repetitividade/precisão	0,10%	0,08%
Tamanho	11,4 cm x 11,4 cm	12,7 cm x 20,3 cm
Tempo de resposta	200 ms	250 ms
Sensibilidade à temperatura	Baixa	Alta
Sensibilidade à pressão	Alta	Baixa

Fonte: adaptado de (SHEPARD et al., 1993).

Na Tabela 10, foram atribuídos pesos relativos a cada um dos parâmetros da Tabela 9 e, em seguida, computou-se a média geométrica normalizada desses parâmetros.

Tabela 10 - Pesos dos parâmetros dos sensores de pressão.

	Repetitividade/ precisão	Tamanho	Tempo de resposta	Sensibilidade à temperatura	Sensibilidade à pressão	Importância relativa	Normalizada
Repetitividade/ precisão	1	3	2	3	2	2,0476725	0,3575958
Tamanho	0,33	1	0,5	1	0,33	0,55872689	0,09757343
Tempo de resposta	0,5	2	1	1	0,5	0,87055056	0,15202885
Sensibilidade à temperatura	0,5	2	0,5	1	0,33	0,69742384	0,12179481
Sensibilidade à pressão	0,5	2	3	3	1	1,5518455	0,27100700

Fonte: Autores.

Finalmente, a Tabela 11 apresenta a análise de Pugh tomando o sensor capacitivo como referência. O valor "+1" indica que o sensor piezoelétrico tem um desempenho melhor em um determinado parâmetro; "-1" indica um desempenho inferior; e "0" um desempenho equivalente. Essas notas são então ponderadas pelos pesos atribuídos na Tabela 10 e, em seguida, somadas. O resultado é inferior a 0, o que indica que a referência, o sensor capacitivo, é o mais indicado nesta aplicação.

Tabela 11 - Análise de Pugh.

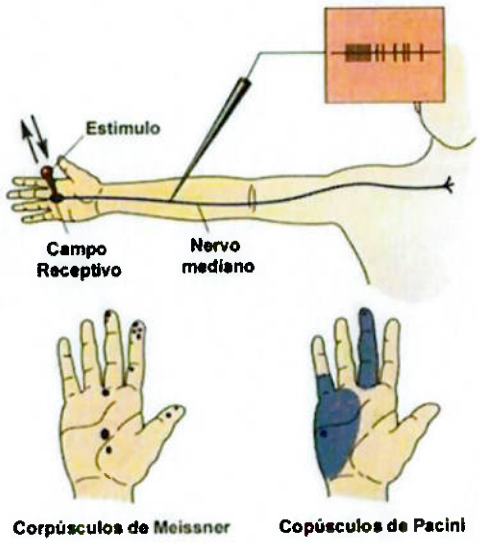
	Capacitivo (referência)	Piezoelétrico	Normalizado
Repetibilidade/acurácia	-	+1	0,357595889
Tamanho	-	-1	-0,09757343497
Tempo de resposta	-	-1	-0,1520288527
Sensibilidade à temperatura	-	-1	-0,1217948167
Sensibilidade à pressão	-	-1	-0,2710070066
Peso	-	-3	**-0,2848082219**

Fonte: Autores.

Os sensores capacitivos foram confeccionados a partir de placas de cobre e de EVA. O EVA atua como o dielétrico e sua constante dielétrica varia entre 2 e 3.

Foram utilizados 2 sensores: 1 na ponta do dedo polegar e outro na ponta do dedo indicador. Em trabalhos preliminares, foi possível comparar o desempenho de diferentes métodos de classificação no tratamento dos dados da base dos sensores táteis do robô *iCub*. Nesses trabalhos, foram obtidos erros de classificação menores que 15% utilizando apenas os dados dos sensores posicionados nestes dois dedos e o método SVM gaussiano (BECARI et al., 2016). Além disso, é nos dedos polegar e indicador que se encontram as maiores concentrações de corpúsculos de Meissner, como ilustrado na Figura 6.

Figura 6 - Corpúsculos de Meissner e Pacini na mão.



Fonte: Adaptado de http://www.ibb.unesp.br/.

Os corpúsculos de Meissner são mecarreceptores táteis responsáveis por parte da capacidade sensorial das mãos, mais especificamente pela detecção de toques rápidos (NISHIDA).

4.4.2 Elemento de conversão

Para poder interpretar os dados provenientes dos sensores, é preciso que os valores de capacitância medidos sejam convertidos em níveis digitais e transmitidos a um computador. A análise de forças e fraquezas dessas duas alternativas é apresentada na Tabela 12.

Tabela 12 - Análise de forças e fraquezas dos elementos conversores.

Elemento conversor	Forças	Fraquezas
Conversor AD	Simplicidade +	Necessidade de cascateamento -- Degradação da relação sinal-ruído ---
CDC	Possui sequenciador automático ++ Apenas um componente +++ Menor degradação da relação sinal-ruído +++	Mais caro - Mais complexo -

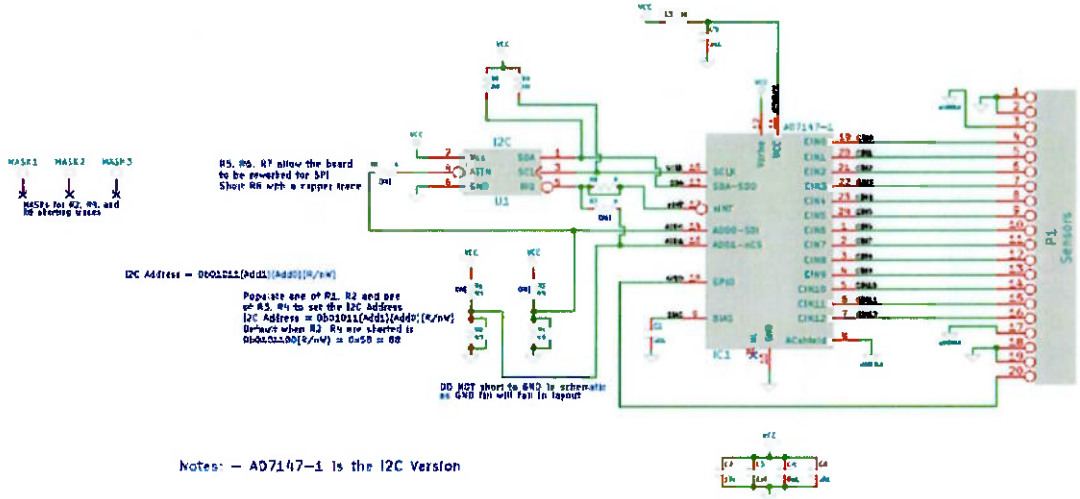
Fonte: Autores.

A opção escolhida foi o CDC, pois exige apenas um componente, reduzindo a possibilidade de degradação do sinal-ruído ou de atenuação do sinal. O CDC utilizado foi o AD7147ACPZ-1500RL7, com encapsulamento SMD, que possui 12 canais de entrada, um sequenciador automático programável (de acordo com as entradas a serem utilizadas) e saída compatível com o protocolo I2C (*Inter-Integrated Circuit*). Os dados são enviados ao computador por meio de uma interface I2C-USB.

Devido ao tamanho do circuito integrado, à ausência de pinos PTH e a uma série de conexões secundárias que precisam ser feitas (blindagens AC, filtro passa-baixas para a tensão de alimentação etc.), não é possível conectar o CDC ao sistema sem antes desenvolver uma placa de circuito impresso satisfazendo aos requisitos de projeto.

Como este CDC é amplamente utilizado em aplicações envolvendo sensores capacitivos de pressão, existem projetos disponíveis em plataformas de compartilhamento. O esquema elétrico que foi utilizado é apresentado na Figura 7. Além de designar soquetes correspondentes aos pinos da interface I2C, à alimentação do CI e às entradas capacitivas, esta arquitetura adiciona um filtro passa-baixas para evitar perturbações na alimentação do circuito, capacitores para evitar oscilações de tensão e blindagens AC, além de apresentar a possibilidade de utilização no modo SPI pela impressão de trilhas que podem ser soldadas em curto-circuito caso desejado.

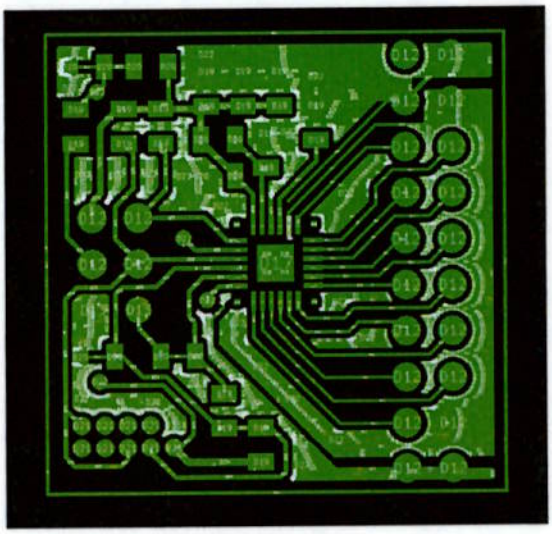
Figura 7 - Esquemático do circuito impresso contendo o AD71478



Fonte: https://github.com/ha7ilm/opendous/tree/master/Current_Designs/AD7147.

A máscara frontal correspondente é apresentada na Figura 8. Esta máscara foi fabricada pela empresa Griffus, especializada em placas de circuito impresso. O AD7147 foi soldado em apenas 5 unidades e, dentre as 5, somente uma foi escolhida para soldar os demais componentes (resistores, capacitores) SMD. Todo o processo descrito, excetuando-se a impressão da placa propriamente dita, foi realizado no Laboratório de Microeletrônica da Escola Politécnica.

Figura 8 - Máscara frontal da placa do AD7147.

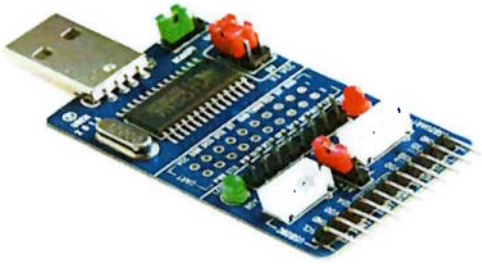


Fonte: Autores.

4.4.3 Elementos de interface

O primeiro módulo de *hardware* adquirido foi a interface *All-in-one*-USB, que possibilita comunicações dos tipos UART, SPI e I2C. Neste projeto, o barramento utilizado é o I2C. Uma foto desta interface é mostrada na Figura 9.

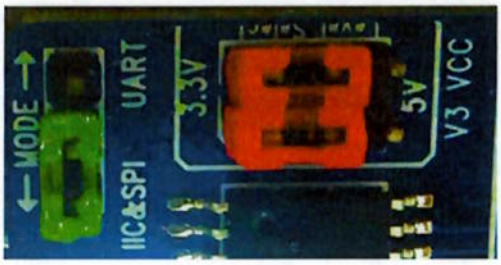
Figura 9 - Interface All-in-one-USB.



Fonte: Amazon.com (2016)

Neste projeto, apenas a saída USB e os pinos I2C (à direita na figura) foram utilizados. *Jumpers* móveis definem o tipo de alimentação (3,3V, que é a utilizada, ou 5V) e o barramento (UART ou I2C/SPI), como mostrado na Figura 10.

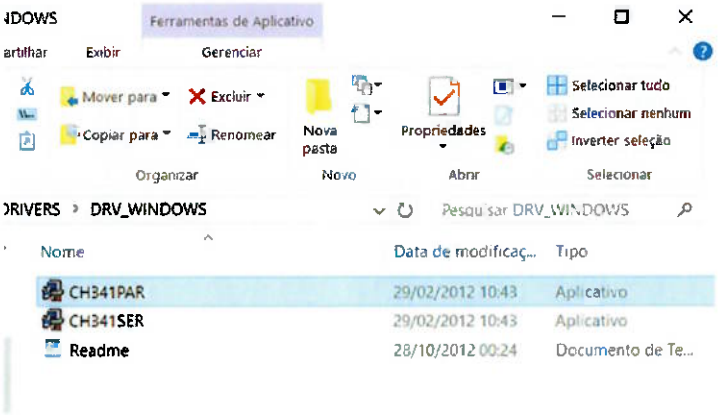
Figura 10 - Detalhe da interface.



Fonte; Autores.

Os *drivers* da interface e alguns programas auxiliares foram fornecidos pelo fabricante Lilly Electronics. É necessário instalar um *driver* para comunicação serial e um para comunicação paralela, como mostrado na Figura 11.

Figura 11 - *Drivers* da interface.



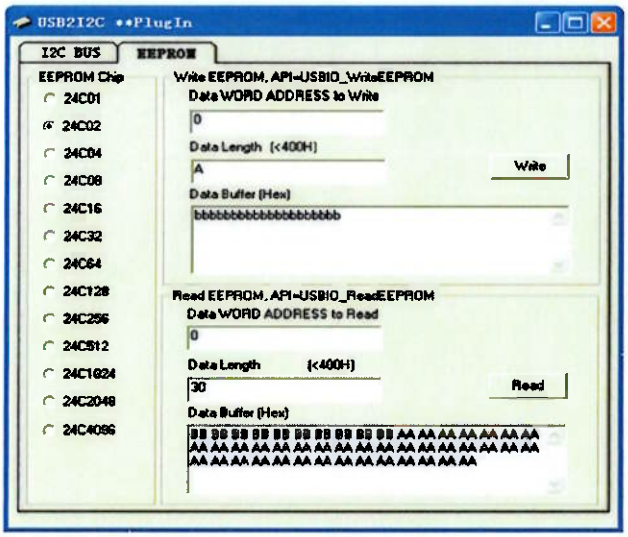
Fonte: Autores.

Após a instalação dos *drivers*, os programas disponíveis (em formato de código e de aplicativo) foram executados para análise de suas funcionalidades e possibilidades de customização. Em particular, a interface do aplicativo para a comunicação I2C é apresentada nas Figuras 12 e 13.

Figura 12 - Programa de leitura/escrita no barramento I2C.

Fonte: Lilly Electronics.

Figura 13 - Programa de leitura/escrita em memórias EEPROM.



Fonte: Lilly Electronics.

Como se pode ver nas figuras, o programa-aplicativo tem duas abas: I2C BUS e EEPROM. A primeira (Figura 12) é utilizada para leitura e escrita em barramentos I2C; a segunda (Figura 13), para realizar as mesmas operações, mas em memórias do tipo EEPROM.

4.4.4 Método de classificação

Em um trabalho preliminar relacionado a este projeto (BECARI et al., 2016), os dados da base dos sensores táteis do robô *iCub* foram classificados utilizando os métodos *complex tree, medium tree, simple tree, linear SVM, quadratic SVM, cubic SVM, gaussian SVM, fine kNN, medium kNN, coarse kNN, cubick NN, boosted trees* e *bagged trees.* A Tabela 13 mostra os resultados obtidos utilizando somente os dados do polegar, do polegar e do indicador, do polegar e do dedo médio e dos três dedos para cada um desses métodos.

Com base nesses resultados, observa-se que o método SVM gaussiano é o mais bem adaptado à classificação de dados provenientes de sensores táteis, pois ele teve o menor erro de classificação em todas as situações em que 2 ou mais dedos são considerados.

A classificação dos dados foi implementada em Matlab, com o uso das bibliotecas *Lib SVM* e *Classification Learner.*

Tabela 13 - Porcentagem de sucesso de diferentes métodos na classificação dos dados dos sensores táteis do robô *iCub*.

Classification Algorithms	*3 fingers (thumb, index and medium)* (%)	*2 fingers (thumb and index)* (%)	*2 fingers (thumb and medium)* (%)	*1 finger (thumb)* (%)
ComplexTree	94.3	81.3	87.5	71.5
Medium Tree	92.4	81.3	86.6	**75.0**
SimpleTree	59.2	55.8	59.4	56.6
Linear SVM	90.8	78.5	85.5	68.4
Quadratic SVM	94.2	83.0	88.6	62.9
Cubic SVM	94.3	81.3	88.0	54.8
Gaussian SVM	**97.4**	**85.1**	**89.0**	74.3
Fine kNN	96.1	82.0	85.8	49.7
Medium kNN	94.6	85.0	88.5	68.4
Coarse kNN	85.3	75.6	83.1	72.1
Cubic kNN	94.4	**85.1**	88.5	68.4
Boosted Trees	94.0	81.3	87.1	74.6
Bagged Trees	95.7	82.8	86.6	70.5

Fonte: BECARI et al., 2016.

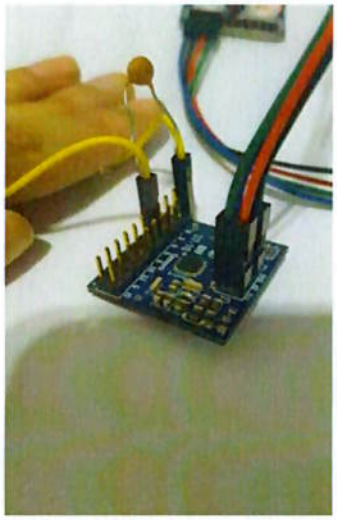
5 Resultados e discussão

Este capítulo apresenta os resultados do projeto. Nele são descritos e apresentados os sensores e a luva desenvolvidos, o *hardware* de conversão CDC, a interface I2C e seu protocolo de configuração e comunicação com o módulo CDC, e, por último, os resultados do *software* para interface com o usuário e classificação, bem como os resultados obtidos na classificação de objetos.

5.1 Elementos de *hardware* desenvolvidos

A placa completamente fabricada é mostrada na Figura 14. Nesta figura, ela também já está conectada à interface por meio dos cabos vermelho, verde, azul e preto e a um capacitor de unidades de picofarads com o auxílio dos soquetes dos cabos amarelos.

Figura 14 - Placa do CDC AD7147.



Fonte: Autores.

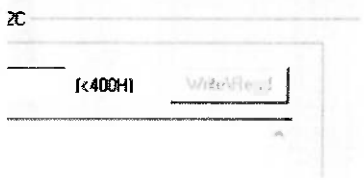
O cabo vermelho liga o V_{DD} do CDC ao V_{DD} da interface; o preto conecta os terras de ambas as placas; o azul conecta os terminais SCK (*clocks*); o verde, os terminais SDA (*serial data*). Nenhuma outra conexão é necessária para a transferência de dados via I2C.

A princípio, nenhum capacitor foi conectado às entradas de capacitância CINx, pois a primeira funcionalidade a ser testada era a comunicação entre: a) o computador e a interface; b) a interface e a placa.

A comunicação entre PC e interface foi verificada pelo estado do botão *Read/Write* na interface do programa I2C *Bus*. Quando a interface não está conectada a nenhuma porta USB do computador, este botão fica inativo, como mostrado na Figura 15.

Para verificar a comunicação entre a interface e o CDC, uma boa alternativa era a escrita e, posteriormente, a verificação pela leitura dos registradores do CDC. Esta etapa requereu o estudo detalhado do *datasheet* do AD7147 para identificar os comandos necessários às operações de leitura e escrita nos registradores do CI.

Figura 15 - O botão *Write/Read* inativo significa que a interface não está conectada ao computador.



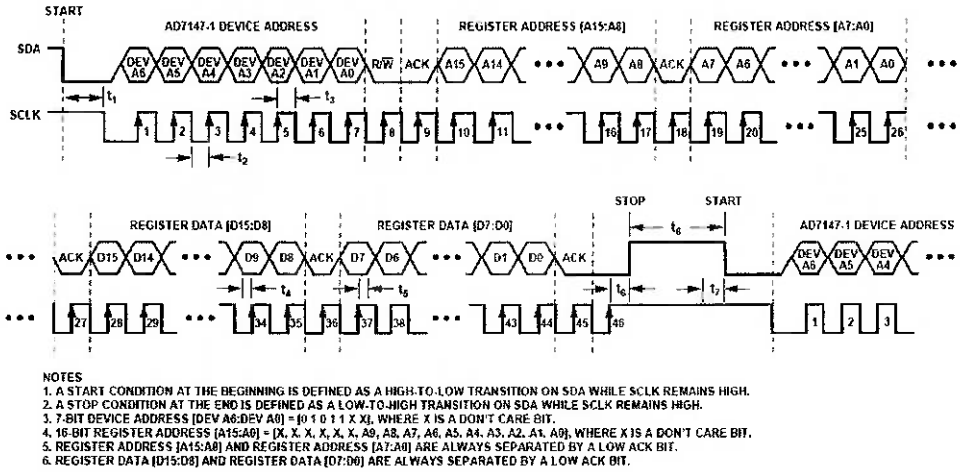
Fonte: Autores.

Todos os registradores do AD7147 (tanto de configuração quanto de dados) são endereçados em 16 bits e contém 16 bits de dados. Quanto à codificação dos dados a serem enviados pelo barramento I2C pelo programa que acompanha a interface, é possível observar na Figura 12 que eles são enviados na forma hexadecimal e que a sua largura em número de *bytes* deve ser informada no campo correspondente à

escrita quando se desejar escrever e no campo correspondente à leitura quando se desejar ler. Assim, dois caracteres hexadecimais correspondem a um *byte*. Como os endereços do CDC são codificados em 16 bits, isto é, em dois *bytes*, foi necessário utilizar 4 caracteres hexadecimais para designá-los.

O protocolo de escrita do AD7147 é apresentado na Figura 16. Ele requer o carregamento do endereço do dispositivo (CDC), de um bit de leitura/escrita, do endereço do registrador e dos dados a serem escritos no barramento.

Figura 16 - Detalhamento do protocolo de escrita I2C no AD7147.



Fonte: *datasheet* do AD7147.

O endereço do dispositivo e o *bit* de leitura/escrita somam 8 bits (1 *byte*). O endereço do registrador e os dados a serem escritos têm 16 bits (ou 2 *bytes*) cada, o que totaliza 5 *bytes* a serem carregados no barramento.

O endereço-base do dispositivo tem 7 bits é dado por 01011XX, onde os dois últimos bits designam os valores dos pinos ADD0 e ADD1. Como se pode ver no esquemático na Figura 7, na placa confeccionada para este projeto ambos os pinos estão fixos em V_{DD} / 2. Assim, o valor de XX no endereço do dispositivo é arbitrário.

Escolheu-se 00; o endereço do dispositivo resulta então 0101100. Concatenando o bit de leitura/escrita, fixo em 0 para indicar a operação de leitura, obtemos o valor do primeiro *byte* a ser carregado no barramento em hexadecimal: 0b01011000 = 0h58.

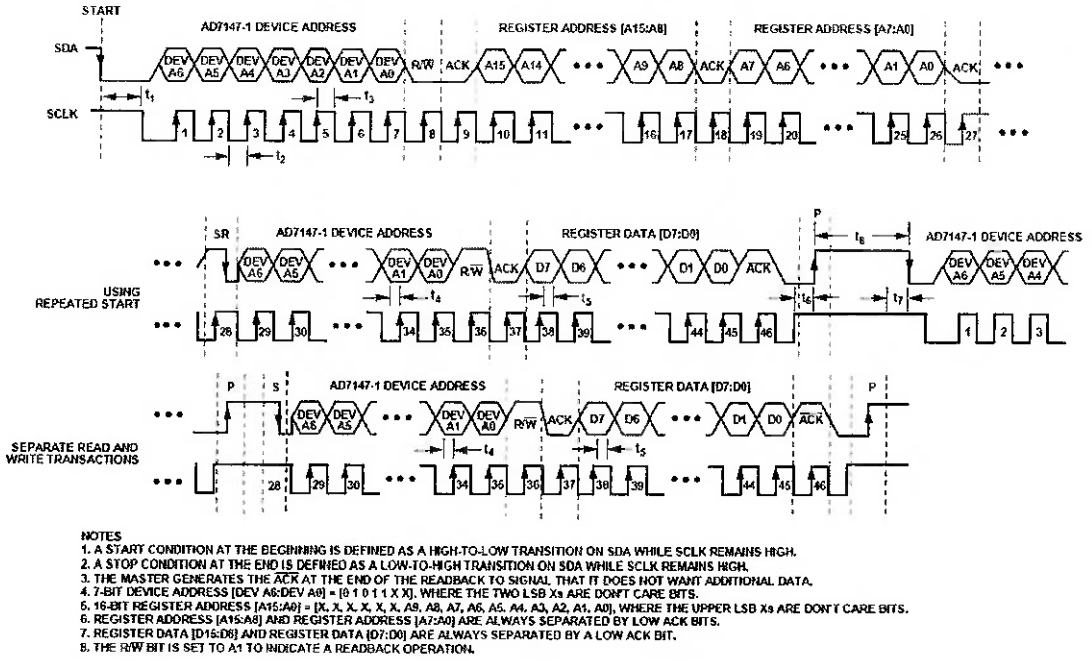
Para simplificar este teste, optou-se por escrever nos *bits* do registrador que controlam o modo de operação do CDC (*full shutdown*, *low power* e *fullpower*). O registrador em questão tem endereço 0x00 e seus dois primeiros bits designam o modo de operação (00 - *full power*, 10 - *low power*, 11 ou 01 - *shutdown*).

Após várias tentativas sem sucesso devido à incorreta compreensão do funcionamento tanto do programa acessório à interface quanto do protocolo de escrita do CDC, foi possível escrever 10 nesses bits realizando as seguintes operações (cf. Figura 12):

- No campo *Data* de *Data to Write*, escreve-se: 5800000002
- No campo *Length* de *Data to Write*: 5 (*bytes*)
- No campo *Length* de *Data to Read*: 0
- Finalmente, pressionamos o botão *Read/Write*

O sucesso da escrita só poderia ser verificado pela leitura do registrador 0x00. Para tal, é preciso dominar o protocolo de leitura dos registradores do AD7147, também detalhado no *datasheet* do componente e apresentado na Figura 17.

Figura 17 - Protocolos de leitura em I2C para o AD7147.



NOTES
1. A START CONDITION AT THE BEGINNING IS DEFINED AS A HIGH-TO-LOW TRANSITION ON SDA WHILE SCLK REMAINS HIGH.
2. A STOP CONDITION AT THE END IS DEFINED AS A LOW-TO-HIGH TRANSITION ON SDA WHILE SCLK REMAINS HIGH.
3. THE MASTER GENERATES THE ACK AT THE END OF THE READBACK TO SIGNAL THAT IT DOES NOT WANT ADDITIONAL DATA.
4. 7-BIT DEVICE ADDRESS [DEV A6:DEV A0] = [0 1 0 1 1 X X], WHERE THE TWO LSB Xs ARE DON'T CARE BITS.
5. 16-BIT REGISTER ADDRESS [A15:A0] = [X, X, X, X, X, X, A9, A8, A7, A6, A5, A4, A3, A2, A1, A0], WHERE THE UPPER LSB Xs ARE DON'T CARE BITS.
6. REGISTER ADDRESS [A15:A8] AND REGISTER ADDRESS [A7:A0] ARE ALWAYS SEPARATED BY LOW ACK BITS.
7. REGISTER DATA [D15:D8] AND REGISTER DATA [D7:D0] ARE ALWAYS SEPARATED BY A LOW ACK BIT.
8. THE R/W BIT IS SET TO A1 TO INDICATE A READBACK OPERATION.

Fonte: *datasheet* do AD7147.

Neste projeto, optou-se por utilizar operações de leitura e escrita separadas para realizar a leitura. Isso significa que toda leitura requer uma escrita prévia - indicando o endereço do dispositivo, o bit de leitura/escrita assinalado em 0 e o endereço do registrador que se deseja ler. Só então a leitura poderá ser realizada, pela indicação do endereço do dispositivo e do bit de leitura/escrita assinalado em 1. A leitura do registrador 0x00 foi efetuada seguindo os passos abaixo:

Escrita do endereço do registrador:

- No campo *Data* de *Data to Write*, escreve-se: 580000
- No campo *Length* de *Data to Write*: 3 (*bytes*)
- No campo *Length* de *Data to Read*: 0
- Pressiona-se o botão *Read/Write*

Leitura do registrador:

- No campo *Data* de *Data to Write*, escreve-se: 59
- No campo *Length* de *Data to Write*: 1 (*byte*)
- No campo *Length* de *Data to Read*: 2 (*bytes*)

O valor lido é 02, que traduzido em binário corresponde a 00000010, o valor escrito anteriormente.

O terceiro teste consiste em ler capacitâncias nos registradores de dados. Para isso, foram providenciados capacitores cerâmicos de 4,7, 5,6 e 6,8 pF, uma vez que o valor máximo de capacitância lido com boa precisão pelo CDC é de 8 pF (apesar de seu *range* atingir algo em torno de 20 pF). Como mostrado na Figura 14, os capacitores foram conectados nos terminais CIN3 e GND. Tentou-se efetuar a leitura do registrador correspondente (0x0E), mas ela retornava zero. Isso ocorreu porque o chip não havia sido configurado corretamente.

De acordo com um exemplo fornecido online pelo fabricante, as configurações necessárias são:

- Registrador 0x00: 0x00B2
- Ler os registradores 0x08, 0x09, 0x0A
- Registrador 0x02: 0x3230
- Registrador 0x03: 0x0819
- Registrador 0x04: 0x0832
- Registrador 0x05: 0x0000
- Registrador 0x06: 0x0000
- Registrador 0x07: 0x0001
- Registrador 0x01: 0x0FFF
- Ler os registradores 0x08, 0x09, 0x0A

Em seguida, deve-se configurar ao menos um dos 12 estágios de leitura do CDC para leitura da capacitância conectada. O estágio escolhido foi o estágio 3, e os capacitores-teste foram conectados à entrada CIN3 e ao GND. As configurações abaixo reúnem estas informações, uma vez que os registradores 98 e 99 são os

registradores de configuração do estágio 3, e que os valores escritos neles especificam que será realizada apenas uma leitura. Esta leitura é positiva (quanto maior a capacitância, maior o valor lido) e deve ser realizada na entrada CIN3.

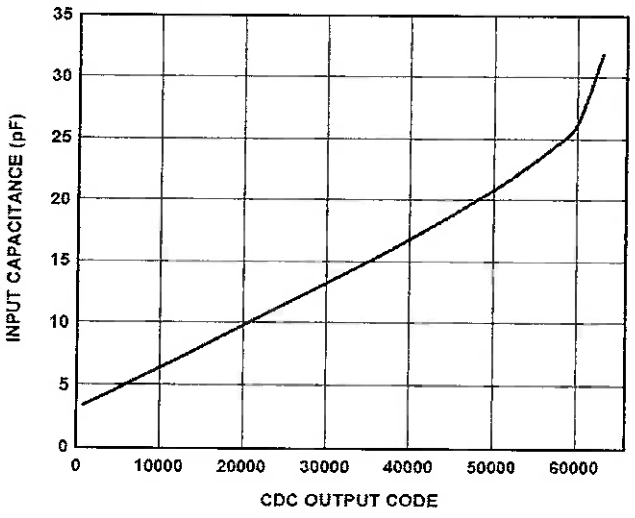
- Registrador 0x98: 0x3FBF
- Registrador 0x99: 0xDFFF

As leituras para cada capacitor são:

- 4,3 pF = F09D (h) = 61597 (d)
- 5,6 pF = F407 (h) = 62471 (d)
- 6,8 pF = FFFF (h) = 65535 (d)

A Figura 18 mostra o gráfico de conversão entre os códigos de leitura (expressos em decimais) e a capacitância de entrada em pF.

Figura 18 - Capacitância de entrada vs. código lido (decimal).



Fonte: *datasheet* do AD7147.

Pela análise deste gráfico, conclui-se facilmente que as leituras de capacitâncias efetuadas apresentam valores maiores do que o esperado (ultrapassando 25 pF),

provavelmente devido às capacitâncias parasitas em paralelo causadas pelos fios, por exemplo. No caso do capacitor de 6,2 pF, pode-se inclusive dizer que a capacidade de leitura do CDC foi ultrapassada.

No entanto, uma constatação positiva é o fato de que as medidas diferem entre si e de que os valores lidos aumentam, proporcionalmente e monotonicamente, com a capacitância. Como neste projeto o interesse principal é medir apenas variações de capacitância, o funcionamento observado é satisfatório.

A quarta etapa consistiu na fabricação do sensor propriamente dito. Ele foi confeccionado a partir de uma lâmina de cobre para as placas paralelas, e de EVA para o dielétrico. O EVA foi escolhido pela facilidade de obtenção do material. Sua constante dielétrica varia entre 2 e 3 e ele é vendido em lâminas de 2 mm de espessura (SEBASTIAN et al., 2015).

Tem-se que:

$$C = \varepsilon . A \div d \quad (1)$$

A partir de (1) obtém-se a área máxima do sensor - aproximadamente 6 cm² -, considerando a constante dielétrica do EVA igual a 3.

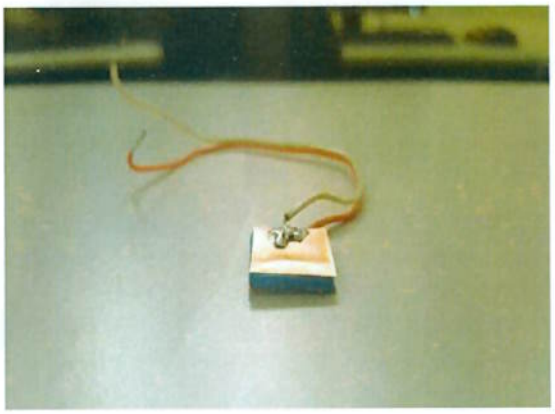
Como o foco deste projeto é a medida de variações de capacitância e a resolução do CDC é de 0,32 pF, deriva-se de (1) a seguinte expressão:

$$\varepsilon . A . \Delta d \div d^2 > 0{,}32\ pF \quad (2)$$

Se for escolhido A = 2 cm², a variação percentual deve ser maior do que aproximadamente 13%.

Uma foto de um sensor fabricado com estas dimensões é apresentada na Figura 19.

Figura 19 - Sensor capacitivo de pressão à base de EVA e cobre.



Fonte: Autores.

Os fios foram soldados nas placas de cobre com estanho, e o EVA foi colado às placas com *Loctite Super Bonder*. A medida de capacitância por meio de um medidor RLC resultou em 5,3 pF para 10kHz e 5,1 pF para 10MHz, o que indica que o valor da capacitância do sensor está estável nesta faixa.

Em seguida, procedeu-se à medida da capacitância do sensor em repouso utilizando o CDC. Os valores lidos ainda foram excessivamente altos; mesmo sem nenhum componente conectado à entrada CIN3, as leituras de capacitância aproximavam-se ou até mesmo ultrapassavam o limite do *range*, resultando em valores próximos de 10 pF.

A partir destas observações, identificou-se a necessidade de uma configuração extra: um ajuste de *offset*. No caso do estágio 3, o registrador que armazena as informações de *offset* é o registrador 9A. Segundo o fabricante, cada bit de *offset* corresponde a um *offset* capacitivo de 0,32 pF. Assim, escolheu-se aplicar um *offset* negativo de 32 bits, isto é, de aproximadamente 10 pF, para anular as capacitâncias parasitas e aquelas dos pinos. O registrador 9A foi então configurado em 00A0.

Desta forma, as leituras para o sensor em repouso resultaram em valores mais próximos do esperado, em torno de 15000 (em decimal) e, portanto, entre 5 e 10 pF. Também foi possível observar aumentos sensíveis de capacitância ao pressionar o sensor; os valores lidos nesses casos não ultrapassaram a faixa dinâmica de leitura do AD7147.

Uma vez verificado o correto funcionamento do sensor capacitivo, foi possível prosseguir com a fabricação do segundo. Ele foi confeccionado nos mesmos moldes do primeiro sensor, porém com uma área um pouco inferior para adaptar-se bem ao dedo indicador. Este sensor foi então conectado à entrada CIN9 e ao GND, e a leitura da entrada 9 foi configurada para ser realizada no estágio 9. Portanto, os dados escritos nos registradores de configuração deste estágio são análogos aos escritos nos registradores do estágio 3.

Alguns dos testes mencionados na seção de Materiais e Métodos não foram necessários - especificamente os relacionados ao tempo de resposta e à resolução de leitura. O tempo de resposta não foi em momento algum um fator limitante, sendo imperceptível para o usuário. Quanto à resolução, mesmo quando pouca pressão é aplicada, esta pressão é detectável pelo CDC, isto é, a resolução dos sensores é menor que a do CDC e, portanto, não caracteriza um fator limitante.

5.2 *Software* de classificação

O sistema de classificação da luva eletrônica e a interface com o usuário constituem a parte de desenvolvimento de *software* deste projeto. Dessa forma, ela se divide duas etapas-chave:

- Desenvolvimento do algoritmo classificador
- Desenvolvimento da interface gráfica

O *app Classification Learner* do Matlab permite importar tabelas contendo *labels* e uma série de dados associada a cada *label*. Esses *labels* são na verdade as variáveis de predição (***x***, que pode ser um vetor de variáveis) e de classificação (***y***, uma única variável), e cada conjunto de valores (*x, y*) fornecidos são amostras de entrada.

Para a luva eletrônica, supõe-se que o número de variáveis de predição e de classificação é o mesmo: as duas variáveis de predição foram as leituras dos sensores dos dedos polegar e indicador, por exemplo; e o resultado será o objeto manipulado. Assim, essa base de dados se adapta bem ao contexto do projeto.

Uma vez importados os dados na biblioteca de classificação, pode-se escolher entre uma série de classificadores: na Figura 20 são mostrados apenas os SVMs quadrático, cúbico, gaussiano fino e médio, mas é possível utilizar também classificadores dos tipos vizinho mais próximo, árvores de decisão e ensemble.

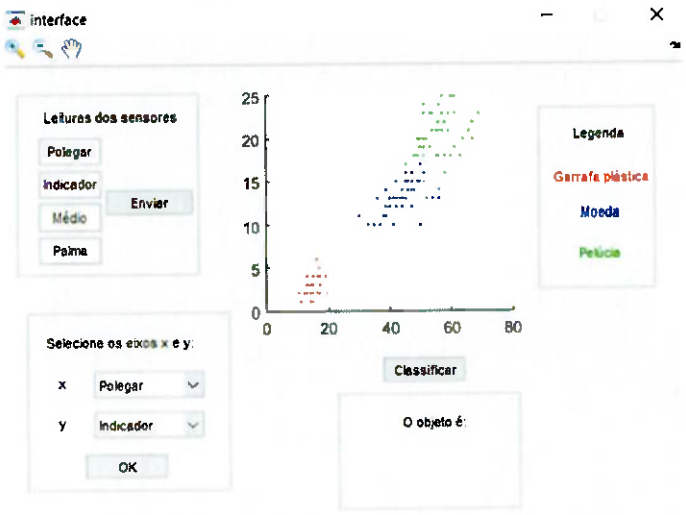
A princípio, o gráfico exibido pela biblioteca mostra uma dispersão das amostras não treinadas, permitindo escolher as variáveis de predição dos eixos *x* e *y*. Após o treinamento, uma segunda informação é mostrada: as amostras mal-classificadas são identificadas por um x ao invés de um ponto, como na Figura 20.

Figura 20 - Biblioteca *Classification Learner* do Matlab após treinamento do classificador SVM médio com 75% das amostras da base *fisheriris*.

Fonte: Autores.

Conforme elucidado nos materiais e métodos, o classificador escolhido foi o SVM gaussiano médio. No que diz respeito ao treinamento/validação do classificador, há três opções: validação cruzada pela criação de partições das amostras e treinamentos múltiplos com cada uma das partições; seleção de uma porcentagem das amostras para treinamento; treinamento com todas as amostras (irrealista). A opção escolhida foi o treinamento com 75% das amostras, que resultou em um classificador com 91,9% de probabilidade de acerto.

Figura 21 - Interface gráfica antes da classificação de uma amostra.



Fonte: Autores.

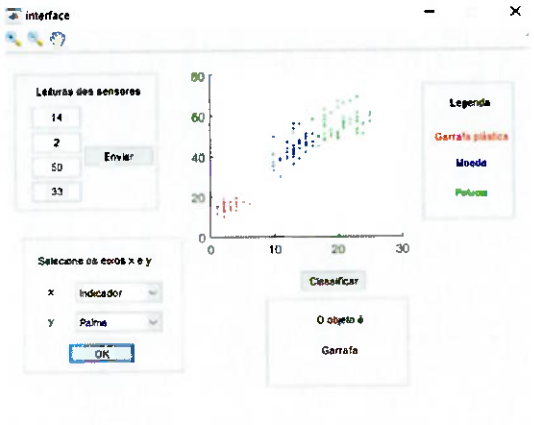
A segunda etapa do desenvolvimento da prova de conceito foi a criação da *GUI* (*Graphical User Interface*) por meio do *GUIDE* (*Graphical User Interface Designer*) do Matlab. Um exemplo de tela da *GUI* desenvolvida é mostrado na Figura 21.

O *GUIDE* é uma ferramenta simples e auto-explicativa. Boa parte da programação é feita pela montagem de blocos como painéis, botões, gráficos, menus *drop-down*, caixas de texto editáveis ou não, etc. Em seguida, é preciso estipular os casos de utilização e o comportamento desses objetos pela escrita de código nas funções *callback* automaticamente criadas pelo Matlab.

A interface *GUI* desenvolvida tem 4 painéis, um botão "Classificar" e um gráfico. O primeiro painel é o de leitura dos sensores, onde os valores das leituras dos sensores do polegar, indicador, médio e da palma devem ser inseridos nas caixas de texto editáveis e enviados pela ativação do botão enviar. O segundo é o painel de seleção dos eixos *x* e *y* do gráfico, escolhidos em listas *drop-down* contendo os valores polegar e indicador. O terceiro painel mostra o resultado da classificação após o envio das leituras dos sensores e a ativação do botão Classificar. O último painel serve apenas como legenda do gráfico de dispersão.

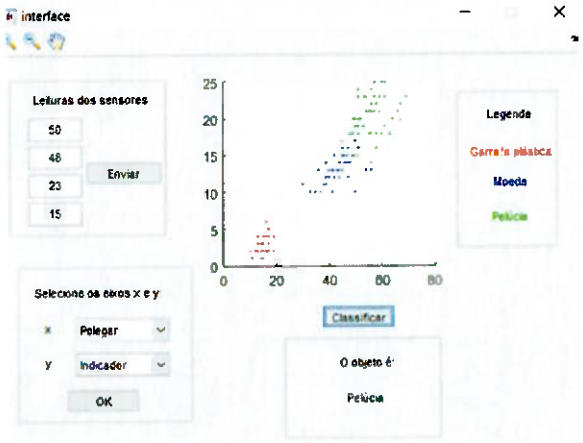
Dois exemplos de classificação diferentes são exibidos nas Figuras 22 e 23.

Figura 22 - Exemplo de leitura dos sensores resultando na classificação em garrafa plástica.



Fonte: Autores.

Figura 23 - Exemplo de leitura dos sensores resultando na classificação em pelúcia.



Fonte: Autores.

5.3 Integração *hardware-software*

A montagem final da luva é mostrada na Figura 24. Os sensores de pressão e o CDC foram colados sobre uma luva fina de poliamida e elastano. Esta luva foi escolhida por ser bastante flexível nos dedos e no punho e, assim, por possibilitar o máximo de adaptabilidade morfológica.

Figura 24 - Montagem final da luva eletrônica.

Fonte: Autores.

Para satisfazer ao requisito de segurança e garantir o correto funcionamento do dispositivo, os sensores foram posteriormente cobertos com fita isolante.

Os próximos passos foram a integração do classificador desenvolvido com toda a parte de *hardware* (interface-CDC-sensores). Embora no modo UART (*Universal Asynchronous Receiver/Transmitter*) a conexão da interface à porta USB crie uma porta serial virtual do tipo COM, facilmente acessível pelo Matlab, no modo I2C o computador enxerga o módulo como uma interface. Além disso, o acesso de periféricos pelas bibliotecas *Instrument Control* e *Data Acquisition* do Matlab só é possível para equipamentos de fabricantes bem específicos, o que não é o caso da interface utilizada.

Uma segunda alternativa era escrever um programa em *C++* ou *Visual Basic* (linguagens em que os programas-exemplo da interface USB-I2C foram disponibilizados) que pudesse ser importado ou executado no Matlab. Explorando a interface I2C-USB e os pacotes associados, procedeu-se à modificação e customização do programa em *Visual Basic 6* (VB6), mais simples do que o equivalente em *Visual C++*, que permite ler e escrever nos registradores do CI conectado à interface como mostrado nas Figuras 12 e 13 da seção anterior.

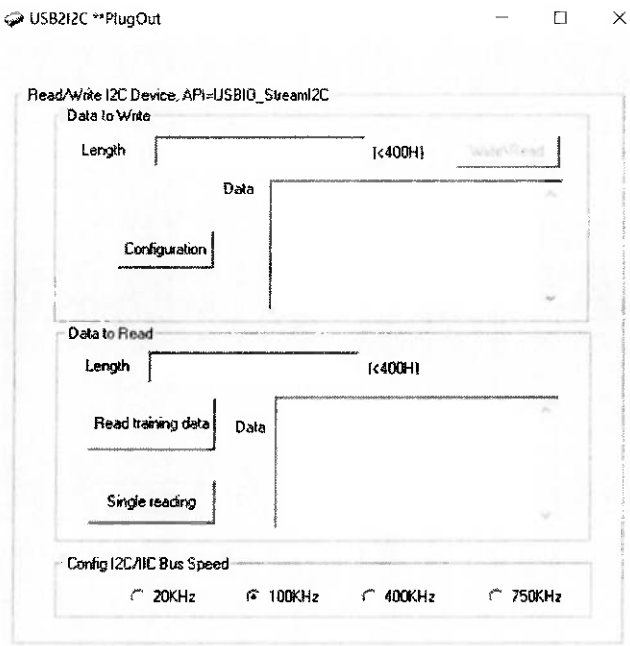
O programa fornecido pelo fabricante precisava ser customizado para simplificação da configuração inicial dos registradores (que precisavam ser escritos um a um) e da leitura de registradores, que necessitava de uma escrita prévia. Além disso, era preciso persistir as leituras dos registradores de resultado em algum tipo de arquivo, tanto para a construção da base de dados do classificador quanto para as leituras únicas realizadas nos momentos da classificação.

Para satisfazer a tais requisitos, a interface e o código do programa em *VB6* foram modificados para incluir tais funcionalidades (cf. Figura 25). O código correspondente pode ser consultado no ANEXO A, com as alterações feitas pelos autores destacadas em azul.

Quando pressionado, o botão *Configuration* realiza todas as escritas nos registradores de configuração listadas na seção 5.1 para os estágios 3 e 9. O botão *Read training data* lê e armazena em dois arquivos texto CIN3.txt e CIN9.txt as leituras de capacitância em CIN3 e CIN9. A cada clique sobre o botão, uma nova leitura é inserida em cada arquivo texto. Leituras sucessivas são separadas por um espaço.

Finalmente, o botão *Single reading* realiza as mesmas operações que *Read training data*, com a exceção de que a cada clique no botão, isto é, a cada nova leitura, o valor previamente escrito no arquivo texto é substituído pela nova leitura.

Figura 25 - Programa da interface I2C-USB modificado.



Fonte: Autores.

Este novo programa permitiu construir bases de dados para teste por meio de cliques sucessivos sobre o botão *Read training data*. Os arquivos .txt gerados poderiam, então, ser lidos pelo Matlab em vetores coluna e utilizados para treinar um novo classificador.

Antes de testar o classificador, havia interesse em identificar a formação de *clusters* para medições realizadas com objetos distintos. Por isso, inicialmente, foram efetuadas 50 medidas de capacitância para 14 objetos: um urso de pelúcia pequeno, um caderno de capa dura, uma garrafa d'água cheia, a mesma garrafa meio cheia, a mesma garrafa vazia, um rolo de fio de estanho laranja, um rolo de fio de estanho azul, uma espuma branca, uma espuma rosa, um pé-de-moleque, um rolo de fita adesiva, um copo plástico cheio, o mesmo copo meio cheio e o mesmo copo vazio. Esses objetos são mostrados nas Figuras 26 e 27.

A base de dados foi lida no Matlab por meio das funções ***fopen*** e ***fscanf*** para manipulação de arquivos. Uma vez salvos nos vetores-coluna com os nomes apropriados, esses dados foram associados a um terceiro vetor coluna contendo números (identificadores) de 0 a 13, cada número identificando um objeto.

Figura 26: Objetos medidos: caderno de capa dura e urso de pelúcia.

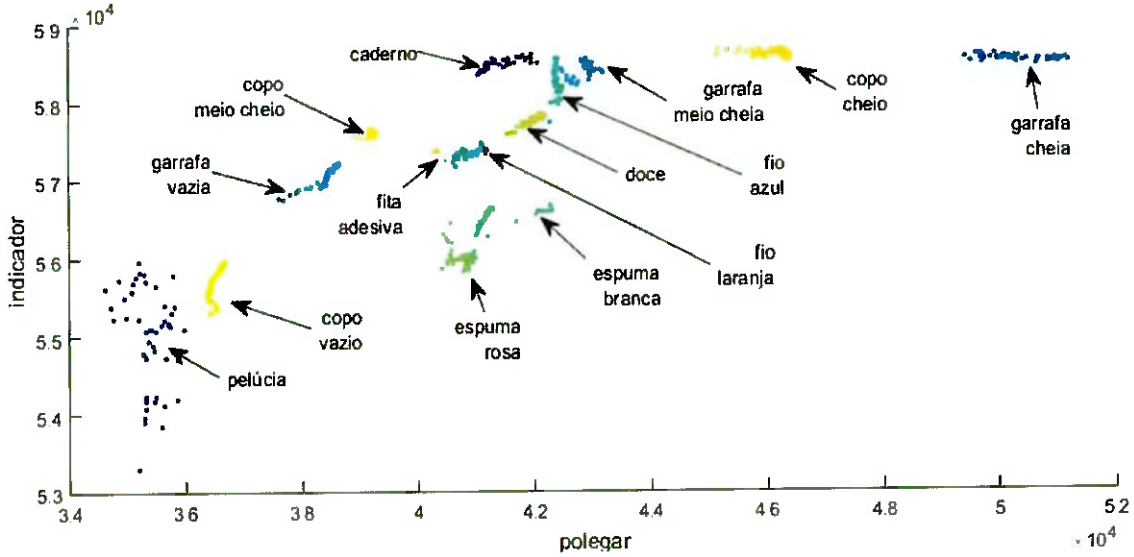
Fonte: Autores.

Figura 27: Demais objetos medidos.

Fonte: Autores.

A análise dos dados obtidos está descrita em um gráfico de dispersão, com códigos de cores para cada objeto, medidas do polegar no eixo *x* e medidas do indicador no eixo *y*. Esse gráfico é mostrado na Figura 28.

Figura 28 - Gráfico de dispersão para os objetos testados.



Fonte: Autores.

A análise desse gráfico mostra que as medidas dos sensores se organizam em *clusters* distinguíveis para cada objeto. No entanto, há sobreposições entre alguns deles (como entre espuma rosa e espuma branca) e outros são muito próximos (fita adesiva e fio laranja), o que pode dificultar a diferenciação entre os objetos nesses casos específicos. No caso das espumas, a sobreposição observada se deve ao fato de os dois objetos serem bastante semelhantes em forma e massa como pode ser observado na Figura 27.

Também é interessante notar que quanto maior a massa do objeto, maiores as leituras dos sensores. O objeto mais pesado é a garrafa d'água cheia, e na Figura 28 pode-se confirmar que as maiores leituras são as correspondentes a esse objeto. Além disso, o mesmo objeto, mas em estados diferentes (copo e garrafa cheios, meio cheios ou vazios) pode ser classificado corretamente de acordo com seu estado, uma vez que os *clusters* copo cheio, meio cheio e vazio estão bem delimitados e distantes entre si. Eles também são crescentes e monotônicos, valendo o mesmo para as garrafas cheia, meio cheia e vazia.

Esses resultados apontam para a possibilidade de utilizar o sistema desenvolvido como um classificador de massa permitindo mensurar as variações de massa de um mesmo objeto ou material.

Também é possível observar que algumas classes têm seus pontos mais concentrados do que outras - é o caso da fita adesiva, por exemplo. Isso de deve ao fato de a fita adesiva ser um objeto rígido. Assim, a pressão aplicada para segurá-la não varia muito, uma vez que o objeto fornece alguma resistência à força aplicada e não se deforma. Já para a pelúcia, as medidas foram mais dispersas, o que reflete sua textura mais mole e deformável. Objetos desse tipo permitem uma aplicação de pressão variável em sua manipulação.

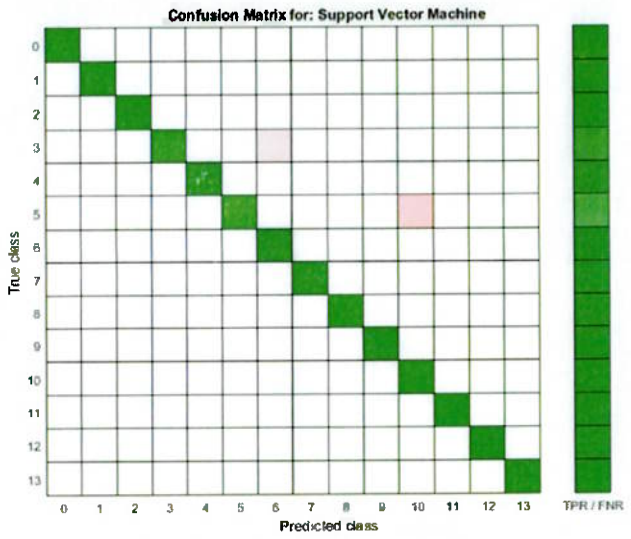
Uma outra observação que pode ser extraída desse gráfico é o fato de os valores medidos no sensor do dedo polegar estarem bem mais dispersos entre si (variando entre 3,4 e 5,2) do que os dados do sensor do dedo indicador (variando entre 5,4 e 5,9). Isso se deve à diferença de área dos dois sensores: o sensor do dedo polegar

é um pouco menor em área do que o do dedo indicador, e, portanto, mais sensível às variações de espessura.

Foi desenvolvido um classificador SVM gaussiano médio com 75% desses dados utilizados no treinamento, a fim de corroborar a hipótese de que, devido à existência de *clusters*, os objetos podem ser classificados dentro de uma margem de erro baixa.

O erro obtido foi de 1,7%, um erro muito menor do que o requisito de 15%. A matriz de confusão correspondente é mostrada na Figura 29.

Figura 29 - Matriz de confusão para o SVM gaussiano médio.



Fonte: Autores.

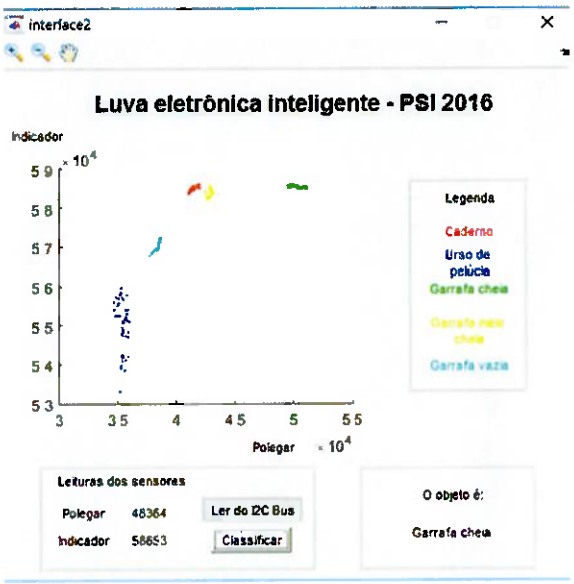
Houve erro de classificação para apenas dois objetos: garrafa meio cheia e fio de estanho laranja. A garrafa meio cheia foi mal classificada como o rolo de fio de estanho azul, provavelmente devido à proximidade entre os pontos medidos para as duas classes. O fio de estanho laranja, por sua vez, foi classificado como fita adesiva. Observando a Figura 28, é possível notar que essas classes também são bastante próximas entre si.

O *GUI* feito em Matlab e apresentado na seção anterior foi modificado para incorporar o novo classificador e ler a partir da porta USB. Este novo *GUI* é mostrado na Figura 30 e seu código está presente no ANEXO C. Para a apresentação dos resultados, optou-se pela utilização de apenas 5 dos 14 objetos das Figuras 26 e 27: caderno, pelúcia e garrafas cheia, meio cheia e vazia.

O classificador obtido a partir do treinamento com 75% desses dados acertou 100% das predições. Como se pode observar na Figura 30, a precisão do SVM para este conjunto de dados não era inesperada, pois as medidas referentes ao caderno, à pelúcia e às garrafas dividem-se em *clusters* bem definidos.

O próximo passo foi garantir que a leitura dos sensores e a classificação do objeto manipulado pela luva ocorressem simultaneamente. Para isso, o programa em VB6 foi alterado (ANEXO B) para que a configuração dos registradores e uma leitura única acontecessem automaticamente e, além disso, que a interface gráfica do programa em VB6 ficasse invisível e que ele encerrasse sua execução logo após a realização das leituras.

Figura 30 - *GUI* adaptado à luva eletrônica.



Fonte: Autores.

O *VB6* permite transformar códigos compilados em aplicativos executáveis (*demos*). Assim, o programa modificado foi transformado em aplicativo e salvo no *workspace* do Matlab. No código do *GUI*, o programa em *VB6* é chamado ao se pressionar o botão Ler I2C Bus (Figura 31) por meio da função *system* do Matlab. A ativação deste botão também ativa os *displays* Polegar e Indicador, mostrando o valor das últimas leituras efetuadas em decimal. Estes valores são lidos dos arquivos .txt de leitura única gerados e modificados pelo programa feito em *VB6*. A classificação só ocorre após um clique sobre o botão Classificar - é no *callback* deste botão que é realizada a predição com base nos valores lidos e no classificador carregado no *workspace*.

As manipulações da pelúcia, da garrafa cheia, e da garrafa vazia seguidas da leitura do barramento I2C e da classificação resultaram corretas, mesmo mudando-se o ângulo das mãos e os pontos de contato com os objetos. A maioria dos testes para a garrafa meio cheia e o caderno também resultaram na classificação correta do objeto manipulado, mas houve alguns erros isolados devido à proximidade entre as classes. Isso aconteceu principalmente quando os testes foram realizados para um caderno diferente, um pouco mais leve do que o utilizado para treinamento.

6 Conclusão

Neuropatias periféricas afetam uma parcela da população, que passa então a ter sensibilidade reduzida ou até mesmo completamente perdida. As pessoas afetadas por esse tipo de lesão geralmente possuem a capacidade de realizar tarefas como segurar objetos comprometida. A perda de sensibilidade é bastante prejudicial principalmente em se tratando das mãos, uma vez que estas são usadas constantemente para funções de reconhecimento táteis realizadas pelo ser humano.

Dessa forma, o desenvolvimento de instrumentos que visem recuperar essa sensibilidade, ainda que parcialmente, faz-se fundamental para atender as necessidades desta parcela da população. Desenvolver uma luva eletrônica que tenha sensores para simular o sistema nervoso periférico juntamente com um *software* de reconhecimento e classificação de objetos foi o objetivo deste trabalho, tendo sido atingido com sucesso.

A luva é composta por dois sensores acoplados aos dedos indicador e polegar, de forma a permitir que o movimento análogo ao praticado quando do manuseio de uma pinça seja reproduzido, o que facilita a manipulação dos objetos e permite uma leitura eficaz de dados. Estes sensores se encontram conectados a uma placa de processamento dos dados provenientes dos sensores por meio de um circuito integrado AD7147, um componente que converte capacitâncias em níveis digitais.

O princípio de funcionamento escolhido para implementação dos sensores é baseado na variação da capacitância conforme a pressão exercida sobre os eletrodos que compõem sua estrutura. Esse tipo de sensor foi escolhido, pois o CDC interpreta dados de capacitância na entrada e, dado que a finalidade é o reconhecimento tátil de objetos, a pressão necessária para segurá-los adequadamente se faz uma variável adequada para determinar os valores de capacitância. Os sensores foram então construídos com base em dois eletrodos feitos de folha de cobre e que servem de invólucro para uma camada dielétrica de

EVA. Cada um destes sensores foi então conectado a uma das doze possíveis entradas do AD7147 para leitura de capacitância.

Para possibilitar a comunicação entre esta placa contendo os dados obtidos dos sensores e o *software* desenvolvido com apoio do programa *Visual Basic 6* foi necessário interligá-los através de uma interface USB-I2C: os dados processados pela placa contendo o AD7147 são enviados via interface através de um barramento que segue o protocolo de comunicação I2C e estes dados são interpretados pelo *software* criado.

O *software* mencionado foi fornecido pelo fabricante da interface adquirida para a realização da comunicação mencionada. No entanto, este *software* precisou ser adaptado para permitir um funcionamento mais adaptado à finalidade desejada. O AD7147 exige a realização de diversas etapas de configuração que antecedem sua operação, o que precisou ser implementado e automatizado.

Outro programa criado para interpretar os níveis digitais convertidos pelo AD7147 diz respeito a um *software* de classificação. Usando a biblioteca do Matlab foi possível programar uma interface gráfica que classifica os dados de leitura de um determinado objeto em *clusters* que são identificados todas as vezes que os sensores fazem leituras correspondentes aos dados observados nestes *clusters*. Estes *clusters* foram formados a partir da realização de 50 leituras para cada objeto considerado, sendo suficiente para permitir uma classificação adequada. O método selecionado para interpretar os dados atinge taxa de acertos de 98,3% na classificação para 14 objetos, o que confere alta confiabilidade ao procedimento.

Tabela 14 - Requisitos de usuário e validação.

Requisito	Validado?
A luva eletrônica deve ser confortável e ergonômica: tem de se encaixar bem à mão, permitindo que ela se movimente e manipule objetos sem incomodar o usuário. Os cabos ligando a mão ao computador devem ter o tamanho adequado para que o usuário não tenha seus movimentos restringidos.	SIM. Luva de poliamida e elastano.
O material da luva deve ser inerte e isolante, para que o usuário não desenvolva irritações ou corra risco de choque elétrico.	SIM. O material da luva é isolante e usa-se fita isolante para proteger os sensores.
O tempo de resposta do sistema de sensores, de transmissão e de classificação deve ser o menos perceptível possível para o usuário, preferencialmente, inferior ao tempo de reação humano médio, de 275 ms (HUMAN BENCHMARK).	SIM. O tempo de resposta é imperceptível e não afeta o uso do sistema pelo usuário.
A interface de usuário do *software* deve ser clara e compreensível para usuários leigos, sem conhecimentos em eletrônica ou métodos de classificação.	SIM. A interface do programa é auto-explicativa e 3 usuários confirmaram que ela é clara e fácil de usar.

Fonte: Autores.

Tabela 15 - Restrições ambientais e validação.

Requisito	Validado?
A temperatura de operação não deve afetar o comportamento de nenhum dos circuitos integrados (para o CDC AD7147, a temperatura de operação deve estar entre -40 e 105 graus Celsius) (ANALOG DEVICES) nem alterar significativamente a constante dielétrica ou as propriedades físicas (forma, espessura do dielétrico na ausência de pressão) dos sensores capacitivos.	SIM. Não há alterações de temperatura perceptíveis.

Fonte: Autores.

Tabela 16 - Requisitos técnicos e validação.

Requisito	Validado?
O comprimento dos cabos ligando os sensores da luva ao computador não pode ser excessivo para evitar a atenuação de sinal além de um valor limite.	SIM. Os cabos ligando tanto os sensores ao CDC quanto o CDC à interface são curtos e finos e não há degradação da relação sinal-ruído ou atenuação em nível prejudicial ao correto funcionamento do sistema.
A conversão AD dos sinais provenientes dos sensores deve ser feita o mais próximo possível dos sensores para evitar a degradação da relação sinal-ruído além de um valor limite.	
O erro do classificador deve ser menor do que 15% e o conjunto de treinamento não pode ser grande, a fim de evitar *overfitting*.	SIM. O erro do classificador para 14 objetos variados é inferior a 2%. Para a pelúcia, o caderno e a garrafa em diferentes estados apenas, não houve erro de treinamento.

Fonte: Autores.

Observando os resultados é possível concluir que de fato o sistema implementado atende aos requisitos estabelecidos durante a concepção e o planejamento do projeto. Os requisitos são reapresentados nas Tabelas 14 a 16, que também discorrem sobre sua validação ou não.

O protótipo completo, considerando *hardware e software*, foi desenvolvido conforme planejado e a identificação do objeto manuseado é feita facilmente e em tempo real. Além disso, a maior parte dos conhecimentos adquiridos no curso de Engenharia Elétrica com ênfase em Sistemas Eletrônicos foi muito útil e em alguns casos indispensável à realização deste projeto. Entre os conceitos aprendidos utilizados, podem ser mencionados: o projeto de placas de circuitos integrados; protocolos de comunicação serial; codificação hexadecimal; funcionamento de sistemas digitais síncronos; memórias; desenvolvimento de *software* em diversas plataformas; desenvolvimento e gestão de um projeto em engenharia. Portanto, este projeto de formatura representa um boa síntese dos conteúdos aprendidos nos últimos anos e um fechamento para o curso de Sistemas Eletrônicos.

Algumas dificuldades a serem relatadas estão relacionadas à adaptação do *software* obtido através do fabricante da interface USB-I2C. A dificuldade inicial encontrada estava associada ao fato de que os códigos fornecidos e a interface gráfica do programa tinham explicações em chinês, o que a princípio impossibilitou o entendimento sobre o funcionamento da interface. A segunda dificuldade está atrelada ao uso do programa *Visual Basic 6*, uma vez que este programa nunca havia sido usado anteriormente pelos integrantes do projeto. A adaptação do código demandou este aprendizado para que as tarefas de integração entre *hardware* e *software* pudessem ser concluídas.

Sendo assim, pode-se dizer que, mesmo que a realização desta luva eletrônica inteligente tenha permitido o emprego de conhecimentos já dominados pelos integrantes do projeto, ela também apresentou desafios com os quais foi necessário lidar ao longo de toda a implementação. Sem dúvida alguma, trabalhar com o circuito integrado AD7147 foi o maior dos desafios, uma vez que este foi o primeiro contato tido com o componente, cujas configurações e operação são complexas. Além disso, outros pontos desafiadores se apresentam no que tange o uso de ferramentas de *software* antes desconhecidas, pouco ou até mesmo nunca usadas anteriormente. A superação destes desafios e o cumprimento dos requisitos de projeto estabelecidos garantiram o sucesso na implementação e a entrega de uma ferramenta de instrumentação biomédica e tecnologia assistida de baixo custo que facilitará a realização de atividades cotidianas para pessoas com neuropatias periféricas.

Mediante as modificações necessárias, o sistema desenvolvido também é passível de utilização em outras aplicações, como classificação de alimentos, identificação de ferramentas defeituosas e diagnósticos em engenharia biomédica.

Como observação final e sugestão para trabalhos futuros, é importante notar que a luva desenvolvida constitui apenas um sistema de sensoriamento. Portanto, ela não pode ser empregada como uma prótese devido à inexistência de interação entre a luva e o sistema nervoso do usuário. Para que tal utilização seja possível, vislumbra-

se o desenvolvimento de uma malha de atuação controlada pelo sistema desenvolvido neste projeto.

Referências

AD7147-1_Capacitive_Touch_Sensor-Schematic Disponível em:<https://github.com/ha7ilm/opendous/tree/master/Current_Designs/AD7147> Acesso em: 12.08.2016.

ANALOG DEVICES. Datasheet: AD7147. CapTouch Programmable Controller for Single-Electrode Capacitance Sensors. Disponível em: <http://www.analog.com/media/en/technical-documentation/data-sheets/AD7147.pdf>. Acesso em: 24.05.2016.

BECARI, W. et al. Comparative Analysis of Classification Algorithms on Tactile Sensors, In: PROCEEDINGS AT INTERNATIONAL SYMPOSIUM ON CONSUMER ELECTRONICS (ISCE) 2016. **Proceedings...** 2016.

BRYCHTA, M. Measure capacitive sensors with a sigma-delta modulator. **Electronic Design**, 2005. Disponível em: <http://electronicdesign.com/analog/measure-capacitive-sensors-sigma-delta-modulator>. Acesso em: 27.06.2016.

Classification Trees. **Statsoft Inc** .Disponível em: <http://www.fmi.uni-sofia.bg/fmi/statist/education/textbook/eng/stclatre.html>. Acesso em: 09.05.2016.

CROWDER, R. M. Automation and Robotics: tactile sensing, 1998. Disponível em: <http://www.southampton.ac.uk/~rmc1/robotics/artactile.htm>. Acesso em: 29.03.2016.

HERBRICH, R. **Learning kernel classifiers: theory and algorithms**. Boston: The MIT Press, 2002.

HOSHINO, M. Neuropatias: o que é neuropatia periférica? **SNC Neurologia**. Disponível em: <http://www.sncneurologia.com.br/neuropatia-periferica.htm>. Acesso em: 09.05.2016.

Human Benchmark. Disponível em: <http://www.humanbenchmark.com/tests/reactiontime/statistics>. Acesso em: 24.05.2016.

INOVAÇÃO TECNOLÓGICA. Luva eletrônica amplifica sensibilidade de cirurgiões. ***Inovação Tecnológica***, 2012. Disponível em: <http://www.inovacaotecnologica.com.br/noticias/noticia.php?artigo=luva-eletronica&id=010110120811#.V0TGyZErKhd>. Acesso em: 22.05.2016.

KIM, D. H. et al. Epidermal Electronics. **Science**, v. 333, p. 838-843, 2011.

KO, K. H., WANG, Q. Touch mode capacitive pressure sensors. **Sensors and Actuators A: Physical**, v. 75, p. 242-251, 1999.

MEYER, D. Support Vector Machines, 2009. Disponível em: <http://cran.rproject.org/web/packages/e1071/vignettes/svmdoc.pdf>. Acesso em: 09.05.2016.

NISHIDA, S. M. Sentindo o mundo através da somestesia: o tato. **Como funciona o corpo humano?** Disponível em: <http://www2.ibb.unesp.br/Museu_Escola/2_qualidade_vida_humana/Museu2_qualidade_corpo_sensorial_somestesia1.htm>. Acesso em: 9 Maio 2016.

PUERS, R. Capacitive sensors: when and how to use them. **Sensors and Actuators A: Physical**, v. 37-38, p. 93-105, 1993.

REIS, M. C. et al. Desenvolvimento de uma palmilha instrumentada para pé diabético. In: CONGRESSO BRASILEIRO DE AUTOMÁTICA, 18. **Anais...** Bonito-MS, 12 a 16 Setembro, 2010.

ROMAN, R., VIDAL, T. B. Neuropatias periféricas: dos sintomas ao diagnóstico e tratamento. **MedicinaNET**, 2013.Disponível em: <http://www.medicinanet.com.br/conteudos/revisoes/5373/neuropatias_perifericas.htm>. Acesso em: 09.05.2016.

SCHMITZ, A. et al. A Tactile Sensor for the Fingertips of the Humanoid Robot iCub. In: PROCEEDINGS AT THE IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS. **Proceedings...** Taipei, Taiwan, 2010.

SCHÖLKOPF, B. et al. **Learning with Kernels: support vector machines, regularization, optimization, and beyond**. Local: The MIT Press, 2002.

SCHOTT, G. D. Penfield's homunculus: a note on cerebral cartography. **Journal of Neurology, Neurosurgery and Psychiatry**, v. 56, p. 329-33, 1993.

SCHWARTZ, G. et al. Flexible polymer transistors with high pressure sensitivity for application in electronic skin and health monitoring. **Nature Communications**, v. 4, p. 1859, 2013. doi:10.1038/ncomms2832.

SEBASTIAN, J.; et al. Enhancement in the Electrical and Thermal Properties of Ethylene Vinyl Acetate (EVA) Co-Polymer by Zinc Oxide Nanoparticles. **Open Journal of Composite Materials**, v. 5, p. 79-91, 2015. doi: 10.4236/ojcm.2015.53011.

SHEPARD, R. L., THACKER, L. H. **Evaluation of Pressure Sensing Concepts: A Technology Assessment**. Oak Ridge National Laboratory, Oak Ridge, 1993.

SRINIVASAN, M. A., BASDOGAN, C. Haptics in virtual environments: Taxonomy, research status and challenges. **Computers & Graphics**, Great Britain, v. 21, n. 4, p. 393-404, 1997.

WANG, C. et al. User-interactive electronic skin for instantaneous pressure visualization. **Nature Materials**, v. 12, p. 899-904, 2013.

WU, X. et al. Top 10 algorithms in data mining. **Knowledge Information Systems**, v. 14, p. 1-37, 2008.

ANEXO A - Código para treinamento em VB6

Form frmMain

```
Option Explicit

Dim hopen As Long

Private Sub Command1_Click()

  Dim iBuff As arrRBuffer
  Dim buffer As arrRBuffer

  'mWRLen = HexToBcd(I2CWRLen.Text)
  'mRdLen = HexToBcd(I2CRDLen.Text)

  'Registrador 0
  Call mStrtoVal("58000000B2", buffer, "5")      '½«ÊäËëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

  If (mOpen = True) Then
   If (USBIO_StreamI2C(mIndex, "5", buffer, "0", iBuff) = False) Then
     MsgBox "Erro na escrita do registrador 0", vbExclamation, "USB2I2C DEMO"
   End If
  End If

  'Leitura do registrador 8
  Call mStrtoVal("580008", buffer, "3")      '½«ÊäËëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

  If (mOpen = True) Then
   If (USBIO_StreamI2C(mIndex, "3", buffer, "0", iBuff) = False) Then
     MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
   End If
  End If



  Call mStrtoVal("59", buffer, "1")      '½«ÊäËëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

  If (mOpen = True) Then
   If (USBIO_StreamI2C(mIndex, "1", buffer, "2", iBuff) = False) Then
     MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
   Else

      Dim buff As String
      Dim i As Long
      For i = 0 To 2 - 1
       buff = buff & Hex2bit(iBuff.buf(i)) + " "
      Next
      I2CRDBuf.Text = buff

   End If
  End If
```

```
'Leitura do registrador 9
Call mStrtoVal("580009", buffer, "3")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
 If (USBIO_StreamI2C(mIndex, "3", buffer, "0", iBuff) = False) Then
   MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
 End If
End If



Call mStrtoVal("59", buffer, "1")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
 If (USBIO_StreamI2C(mIndex, "1", buffer, "2", iBuff) = False) Then
   MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
 Else

    For i = 0 To 2 - 1
     buff = buff & Hex2bit(iBuff.buf(i)) + " "
    Next
    I2CRDBuf.Text = buff

 End If
End If

'Leitura do registrador A
Call mStrtoVal("58000A", buffer, "3")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
 If (USBIO_StreamI2C(mIndex, "3", buffer, "0", iBuff) = False) Then
   MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
 End If
End If



Call mStrtoVal("59", buffer, "1")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
 If (USBIO_StreamI2C(mIndex, "1", buffer, "2", iBuff) = False) Then
   MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
 Else

    For i = 0 To 2 - 1
     buff = buff & Hex2bit(iBuff.buf(i)) + " "
    Next
    I2CRDBuf.Text = buff

 End If
End If



'Registrador 2
Call mStrtoVal("5800023230", buffer, "5")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý
```

```
If (mOpen = True) Then
  If (USBIO_StreamI2C(mIndex, "5", buffer, "0", iBuff) = False) Then
    MsgBox "Erro na escrita do registrador 0", vbExclamation, "USB2I2C DEMO"
  End If
End If

'Registrador 3
Call mStrtoVal("5800030819", buffer, "5")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
  If (USBIO_StreamI2C(mIndex, "5", buffer, "0", iBuff) = False) Then
    MsgBox "Erro na escrita do registrador 0", vbExclamation, "USB2I2C DEMO"
  End If
End If

'Registrador 4
Call mStrtoVal("5800040832", buffer, "5")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
  If (USBIO_StreamI2C(mIndex, "5", buffer, "0", iBuff) = False) Then
    MsgBox "Erro na escrita do registrador 0", vbExclamation, "USB2I2C DEMO"
  End If
End If

'Registrador 5
Call mStrtoVal("5800050000", buffer, "5")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
  If (USBIO_StreamI2C(mIndex, "5", buffer, "0", iBuff) = False) Then
    MsgBox "Erro na escrita do registrador 0", vbExclamation, "USB2I2C DEMO"
  End If
End If

'Registrador 6
Call mStrtoVal("5800060000", buffer, "5")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
  If (USBIO_StreamI2C(mIndex, "5", buffer, "0", iBuff) = False) Then
    MsgBox "Erro na escrita do registrador 0", vbExclamation, "USB2I2C DEMO"
  End If
End If

'Registrador 7
Call mStrtoVal("5800070001", buffer, "5")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
  If (USBIO_StreamI2C(mIndex, "5", buffer, "0", iBuff) = False) Then
    MsgBox "Erro na escrita do registrador 0", vbExclamation, "USB2I2C DEMO"
  End If
End If

'Registrador 1
Call mStrtoVal("5800010FFF", buffer, "5")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
  If (USBIO_StreamI2C(mIndex, "5", buffer, "0", iBuff) = False) Then
```

```
    MsgBox "Erro na escrita do registrador 0", vbExclamation, "USB2I2C DEMO"
  End If
End If

'Leitura do registrador 8
Call mStrtoVal("580008", buffer, "3")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
  If (USBIO_StreamI2C(mIndex, "3", buffer, "0", iBuff) = False) Then
    MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
  End If
End If



Call mStrtoVal("59", buffer, "1")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
  If (USBIO_StreamI2C(mIndex, "1", buffer, "2", iBuff) = False) Then
    MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
  Else

      For i = 0 To 2 - 1
        buff = buff & Hex2bit(iBuff.buf(i)) + " "
      Next
      I2CRDBuf.Text = buff

  End If
End If



'Leitura do registrador 9
Call mStrtoVal("580009", buffer, "3")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
  If (USBIO_StreamI2C(mIndex, "3", buffer, "0", iBuff) = False) Then
    MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
  End If
End If



Call mStrtoVal("59", buffer, "1")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
  If (USBIO_StreamI2C(mIndex, "1", buffer, "2", iBuff) = False) Then
    MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
  Else

      For i = 0 To 2 - 1
        buff = buff & Hex2bit(iBuff.buf(i)) + " "
      Next
      I2CRDBuf.Text = buff

  End If
```

```
End If

'Leitura do registrador A
Call mStrtoVal("58000A", buffer, "3")      '½«ÊäËëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
 If (USBIO_StreamI2C(mIndex, "3", buffer, "0", iBuff) = False) Then
   MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
 End If
End If


Call mStrtoVal("59", buffer, "1")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
 If (USBIO_StreamI2C(mIndex, "1", buffer, "2", iBuff) = False) Then
   MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
 Else

    For i = 0 To 2 - 1
     buff = buff & Hex2bit(iBuff.buf(i)) + " "
    Next
    I2CRDBuf.Text = buff

 End If
End If

'Configurando os registrados de CIN3

'Registrador 98
Call mStrtoVal("5800983FBF", buffer, "5")      '½«ÊäËëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
 If (USBIO_StreamI2C(mIndex, "5", buffer, "0", iBuff) = False) Then
   MsgBox "Erro na escrita do registrador 0", vbExclamation, "USB2I2C DEMO"
 End If
End If

'Registrador 99
Call mStrtoVal("5800999FFF", buffer, "5")      '½«ÊäËëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
 If (USBIO_StreamI2C(mIndex, "5", buffer, "0", iBuff) = False) Then
   MsgBox "Erro na escrita do registrador 0", vbExclamation, "USB2I2C DEMO"
 End If
End If

'Registrador 9A
Call mStrtoVal("58009A00A0", buffer, "5")      '½«ÊäËëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
 If (USBIO_StreamI2C(mIndex, "5", buffer, "0", iBuff) = False) Then
   MsgBox "Erro na escrita do registrador 0", vbExclamation, "USB2I2C DEMO"
 End If
End If
```

```
'Configurando os registrados de CIN9

'Registrador C8
Call mStrtoVal("5800C83FFF", buffer, "5")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
  If (USBIO_StreamI2C(mIndex, "5", buffer, "0", iBuff) = False) Then
    MsgBox "Erro na escrita do registrador 0", vbExclamation, "USB2I2C DEMO"
  End If
End If

'Registrador C9
Call mStrtoVal("5800C99FEF", buffer, "5")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
  If (USBIO_StreamI2C(mIndex, "5", buffer, "0", iBuff) = False) Then
    MsgBox "Erro na escrita do registrador 0", vbExclamation, "USB2I2C DEMO"
  End If
End If

'Registrador CA
Call mStrtoVal("5800CA00A0", buffer, "5")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
  If (USBIO_StreamI2C(mIndex, "5", buffer, "0", iBuff) = False) Then
    MsgBox "Erro na escrita do registrador 0", vbExclamation, "USB2I2C DEMO"
  End If
End If

End Sub


Private Sub Command2_Click()

Dim iBuff As arrRBuffer
Dim buffer As arrRBuffer
'Leitura do registrador E
  Call mStrtoVal("58000E", buffer, "3")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

  If (mOpen = True) Then
    If (USBIO_StreamI2C(mIndex, "3", buffer, "0", iBuff) = False) Then
      MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
    End If
  End If




  Call mStrtoVal("59", buffer, "1")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

  If (mOpen = True) Then
    If (USBIO_StreamI2C(mIndex, "1", buffer, "2", iBuff) = False) Then
      MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
    Else

        Dim buff As String
        Dim i As Long
```

```
        For i = 0 To 2 - 1
          buff = buff & Hex2bit(iBuff.buf(i)) + ""
        Next
        I2CRDBuf.Text = buff
        Dim intFile As Integer
        Dim strFile As String
        strFile = "C:\Users\Public\CIN3.txt"
        intFile = FreeFile
        Open strFile For Input As #intFile
        Dim FiletoStr As String
        FiletoStr = StrConv(InputB(LOF(intFile), intFile), vbUnicode)
        Close #intFile
        Open strFile For Output As #intFile
        Print #intFile, FiletoStr + CStr(HexToDec(buff)) + " ";
        Close #intFile
      End If
    End If

    'Leitura do registrador 14
    Dim iBuff2 As arrRBuffer
    Dim buffer2 As arrRBuffer

    Call mStrtoVal("580014", buffer2, "3")      '½«ÊäËëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

    If (mOpen = True) Then
      If (USBIO_StreamI2C(mIndex, "3", buffer2, "0", iBuff2) = False) Then
        MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
      End If
    End If



    Call mStrtoVal("59", buffer2, "1")      '½«ÊäËëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

    If (mOpen = True) Then
      If (USBIO_StreamI2C(mIndex, "1", buffer2, "2", iBuff2) = False) Then
        MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
      Else
        Dim buff2 As String
        For i = 0 To 2 - 1
          buff2 = buff2 & Hex2bit(iBuff2.buf(i)) + ""
        Next
        I2CRDBuf.Text = buff2
        strFile = "C:\Users\Public\CIN9.txt"
        intFile = FreeFile
        Open strFile For Input As #intFile
        FiletoStr = StrConv(InputB(LOF(intFile), intFile), vbUnicode)
        Close #intFile
        Open strFile For Output As #intFile
        Print #intFile, FiletoStr + CStr(HexToDec(buff2)) + " ";
        Close #intFile
      End If
    End If
End Sub


Private Sub Command3_Click()
    Dim iBuff As arrRBuffer
```

```
Dim buffer As arrRBuffer
'Leitura do registrador E
  Call mStrtoVal("58000E", buffer, "3")      '½«ÊäËëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

  If (mOpen = True) Then
   If (USBIO_StreamI2C(mIndex, "3", buffer, "0", iBuff) = False) Then
     MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
   End If
  End If



  Call mStrtoVal("59", buffer, "1")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

  If (mOpen = True) Then
   If (USBIO_StreamI2C(mIndex, "1", buffer, "2", iBuff) = False) Then
     MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
   Else

      Dim buff As String
      Dim i As Long
      For i = 0 To 2 - 1
       buff = buff & Hex2bit(iBuff.buf(i)) + ""
      Next
      I2CRDBuf.Text = buff
      Dim intFile As Integer
      Dim strFile As String
      strFile = "C:\Users\Luana Ruiz\workspace\CIN3single.txt"
      intFile = FreeFile
      Open strFile For Output As #intFile
      Print #intFile, CStr(HexToDec(buff)) + " ";
      Close #intFile
   End If
  End If

  'Leitura do registrador 14
  Dim iBuff2 As arrRBuffer
  Dim buffer2 As arrRBuffer

  Call mStrtoVal("580014", buffer2, "3")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

  If (mOpen = True) Then
   If (USBIO_StreamI2C(mIndex, "3", buffer2, "0", iBuff2) = False) Then
     MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
   End If
  End If



  Call mStrtoVal("59", buffer2, "1")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

  If (mOpen = True) Then
   If (USBIO_StreamI2C(mIndex, "1", buffer2, "2", iBuff2) = False) Then
     MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
   Else
      Dim buff2 As String
      For i = 0 To 2 - 1
```

```
            buff2 = buff2 & Hex2bit(iBuff2.buf(i)) + ""
          Next
          I2CRDBuf.Text = buff2
          strFile = "C:\Users\Luana Ruiz\workspace\CIN9single.txt"
          intFile = FreeFile
          Open strFile For Output As #intFile
          Print #intFile, CStr(HexToDec(buff2)) + " ";
          Close #intFile
        End If
      End If
    End Sub


Private Sub eepromRdDate_Click()
 Dim mDataAddr As Long
 Dim mLen As Long
 Dim buffer As arrRBuffer
 Dim bu() As Byte
 mLen = HexToBcd(RdDataLen)

 If (RdDataAddr.Text = "") Then
  MsgBox "ÇëÊäÈëÊý¾ÝµÝÔªÆðÊ¼µØÖ·£¡", vbExclamation, "USB2I2C DEMO"
  Exit Sub
 End If
 If (mLen <= 0) Then
  MsgBox "ÇëÊäÈë¶ÁÈ¡³¤¶È£¡", vbExclamation, "USB2I2C DEMO"
  Exit Sub
 End If
 mDataAddr = HexToBcd(RdDataAddr)
 If (mOpen = True) Then
   If (USBIO_ReadEEPROM(mIndex, eepromid, mDataAddr, mLen, buffer)) Then
    Dim buff As String
    Dim i As Long
    For i = 0 To mLen - 1
      buff = buff & Hex2bit(buffer.buf(i)) & " "
    Next i
    RdDataBuf.Text = buff
   Else
    MsgBox "¶ÁE2PROMÊý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
   End If
   RdDataLen.Text = Hex(mLen)
 Else
   MsgBox "USB2I2C Device NOT Oper!", vbExclamation, "USB2I2C DEMO"
 End If
End Sub

Private Sub eepromWrDate_Click()
 Dim mData As Byte
 Dim mDataAddr As Long
 Dim mLen As Long
 Dim buffer As arrRBuffer

 mLen = HexToBcd(WrDataLen.Text)
 If (WrDataAddr.Text = "") Then
  MsgBox "ÇëÊäÈëÊý¾ÝµÝÔªÆðÊ¼µØÖ·£¡", vbExclamation, "USB2I2C DEMO"
  Exit Sub
 End If
 If (mLen <= 0 Or WrDataBuf.Text = "") Then
  MsgBox "ÇëÊäÈëÒªÐ´ÈëµÄÊý¾Ý,³¤¶È£¡", vbExclamation, "USB2I2C DEMO"
  Exit Sub
```

```
 End If

 If (mLen > (Len(WrDataBuf) \ 2)) Then 'ÔÚÊäËë³¤¶È°íÊý¾Ý³¤¶ÈÖÐÈ¡Ð¡Öµ
  mLen = Len(WrDataBuf) \ 2
 End If

 mDataAddr = HexToBcd(WrDataAddr.Text)
 Call mStrtoVal(WrDataBuf.Text, buffer, mLen)  '½«ÊäËëµÄÊ®Áù½øÖÆ,ñÊ½×Ö·ûÊý¾Ý×ª»ÉÊýÖµÊý¾Ý

 If (mOpen = True) Then
   If (USBIO_WriteEEPROM(mIndex, eepromid, mDataAddr, mLen, buffer) = False) Then
     MsgBox "¶ÁE2PROMÊý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
   End If
   WrDataLen.Text = Hex(mLen)
 Else
   MsgBox "USB2I2C Device NOT Oper!", vbExclamation, "USB2I2C DEMO"
 End If
End Sub

Private Sub eppromtype_Click(Index As Integer)
Select Case Index
    Case 0
      eepromid = ID_24C01
    Case 1
      eepromid = ID_24C02
    Case 2
      eepromid = ID_24C04
    Case 3
      eepromid = ID_24C08
    Case 4
      eepromid = ID_24C16
    Case 5
      eepromid = ID_24C32
    Case 6
      eepromid = ID_24C64
    Case 7
      eepromid = ID_24C128
    Case 8
      eepromid = ID_24C256
    Case 9
      eepromid = ID_24C512
    Case 10
      eepromid = ID_24C1024
    Case 11
      eepromid = ID_24C2048
    Case 12
      eepromid = ID_24C4096
 End Select
End Sub

Private Sub Form_Load()
mIndex = 0
   SSTab1.TabVisible(0) = False
   SSTab1.TabVisible(1) = False
   SSTab1.TabVisible(2) = False
   SSTab1.TabVisible(3) = False
   SSTab1.TabVisible(4) = False
   SSTab1.TabVisible(5) = False
hopen = USBIO_OpenDevice(mIndex)
If (hopen = INVALID_HANDLE_VALUE) Then
```

```
    mOpen = False
Else
    mOpen = True
End If
'ÉèÖÃÉè±¸²å°ÎÍ¨Öª
If USBIO_SetDeviceNotify(mIndex, vbNullString, AddressOf mUSBIO_NOTIFY_ROUTINE) = False Then
    MsgBox "ÉèÖÃÉè±¸²å°ÎÍ¨ÖªÊ§°Ü", vbExclamation, "USB2I2C DEMO"
End If
enablebtn (mOpen)

'Executar aqui a configuração inicial
'Executar também a leitura única

Call Command1_Click
Call Command3_Click

'Unload Me

'End

End Sub

Private Sub Form_Unload(Cancel As Integer)
USBIO_SetDeviceNotify mIndex, vbNullString, 0&
If (mOpen = True) Then
    USBIO_CloseDevice (mIndex)

End If
End Sub



Private Sub StreamICRW_Click()
Dim mWRLen As Long
Dim mRdLen As Long
Dim iBuff As arrRBuffer
Dim buffer As arrRBuffer

mWRLen = HexToBcd(I2CWRLen.Text)
mRdLen = HexToBcd(I2CRDLen.Text)

'----------------------------------------
If (I2CM(0).Value = True) Then
    If (USBIO_SetStream(mIndex, &H80) = False) Then
        MsgBox "ÉèÖÃI2CÊ±ÖÓ = 20KHzÊ§°Ü£¡ ", vbExclamation, "USB2I2C DEMO"
        Exit Sub
    End If
ElseIf (I2CM(1).Value = True) Then
    If (USBIO_SetStream(mIndex, &H81) = False) Then
        MsgBox "ÉèÖÃI2CÊ±ÖÓ = 100KHzÊ§°Ü£¡ ", vbExclamation, "USB2I2C DEMO"
        Exit Sub
    End If
ElseIf (I2CM(2).Value = True) Then
    If (USBIO_SetStream(mIndex, &H82) = False) Then
        MsgBox "ÉèÖÃI2CÊ±ÖÓ = 400KHzÊ§°Ü£¡ ", vbExclamation, "USB2I2C DEMO"
        Exit Sub
    End If
ElseIf (I2CM(3).Value = True) Then
    If (USBIO_SetStream(mIndex, &H83) = False) Then
```

```
        MsgBox "ÉèÖÃI2CÊ±ÖÓ = 750KHzÊ§°Ü£¡ ", vbExclamation, "USB2I2C DEMO"
        Exit Sub
      End If
    End If
    '----------------------------------------


    If (mWRLen > 0 And I2CWRBuf.Text = "") Then
     MsgBox "ÇëÊäÈëÒª´µÄÊý¾Ý,³¤¶È£¡", vbExclamation, "USB2I2C DEMO"
     Exit Sub
    End If
    If ((mWRLen = 0) And (mRdLen = 0)) Then
     MsgBox "ÇëÊäÈë¶ÁÊý¾ÝÊùÐèµÄ³¤¶È£¡", vbExclamation, "USB2I2C DEMO"
     Exit Sub
    End If
    If (mWRLen > Len(Trim(I2CWRBuf.Text)) \ 2) Then
      mWRLen = Len(Trim(I2CWRBuf.Text)) \ 2
    End If

    Call mStrtoVal(I2CWRBuf.Text, buffer, mWRLen)      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª»»ÉÊýÖµÊý¾Ý

    If (mOpen = True) Then
     If (USBIO_StreamI2C(mIndex, mWRLen, buffer, mRdLen, iBuff) = False) Then
       MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
     Else
      If (mRdLen > 0) Then   'ÓÐÊý¾Ý·µ»Ø
        Dim buff As String
        Dim i As Long
        For i = 0 To mRdLen - 1
          buff = buff & Hex2bit(iBuff.buf(i)) + " "
        Next
        I2CRDBuf.Text = buff
      End If
     End If
     I2CWRLen.Text = Hex(mWRLen)
     I2CRDLen.Text = Hex(mRdLen)
    Else
     MsgBox "USB2I2C Device NOT Oper!", vbExclamation, "USB2I2C DEMO"
    End If
    End Sub

    Private Sub USBIO_NOTIFY_ROUTINE_KeyUp(KeyCode As Integer, Shift As Integer)  'Éè±¸²å°ÎÍ¨ÖªÀí³ÌÐò
       Dim iEventStatus As Long
       iEventStatus = KeyCode '²å°ÎÊÂ¼þ
       If (iEventStatus = USBIO_DEVICE_ARRIVAL) Then ' Éè±¸²åÈëÊÂ¼þ,ÒÑ¾²åÈë
          If (USBIO_OpenDevice(mIndex) = INVALID_HANDLE_VALUE) Then
             MsgBox "´ò¿ªÉè±¸Ê§°Ü!", vbOK, "USB2I2C DEMO"
             mOpen = False
          Else
             mOpen = True  '´ò¿ª³É¹¦
          End If
       ElseIf (iEventStatus = USBIO_DEVICE_REMOVE) Then ' Éè±¸°Î³öÊÂ¼þ,ÒÑ¾°Î³ö
          USBIO_CloseDevice (mIndex)
          mOpen = False
       End If
       enablebtn (mOpen) 'Éè±¸´ò¿ª,°´Å¥¿ÉÓÃ,Éè±¸Ã»´ò¿ª,°´Å¥½ûÓÃ
    End Sub

    Public Sub enablebtn(ByVal bEnable As Boolean)  'bEnable=true ;¸÷°´Å¥¿ÉÓÃ ;=false:enable:¸÷°´Å¥½ûÓÃ
     With frmMain
```

```
    .StreamICRW.Enabled = bEnable

    .eepromRdDate.Enabled = bEnable
    .eepromWrDate.Enabled = bEnable

    If (bEnable = True) Then '°Ìå±êÌâÏÔÊ¾
       frmMain.Caption = "USB2I2C **PlugIn"

    Else
       frmMain.Caption = "USB2I2C **PlugOut"

    End If
  End With


End Sub
```

Module 1

```
Option Explicit

Type arrRBuffer
   buf(mMAX_BUFFER_LENGTH - 1) As Byte
End Type

Public Const WM_KEYUP = &H101
Public Const BN_CLICK = &H101
Public eepromid As EEPROM_TYPE  'eepromÐÍºÅ
Public Declare Function PostMessage Lib "user32" Alias "PostMessageA" (ByVal hwnd As Long, ByVal wMsg As Long, ByVal
wParam As Long, ByVal lParam As Long) As Long

Public mIndex  As Long
Public mOpen As Boolean


Function HexToDec(str As String) As Long          '½«ÎÄ±¾¿òÖÐÊäÈëµÄÊ®Áù½øÖÆÖµ×ª»»³ÉBCDÂë
Dim Length As Integer
Dim X As String
Dim i As Long
str = Trim(str)
Length = Len(str)
For i = 0 To Length - 1
X = Mid(str, Length - i, 1)
Select Case X
     Case "a", "A"
       HexToDec = HexToDec + 10 * (16 ^ i)
     Case "b", "B"
       HexToDec = HexToDec + 11 * (16 ^ i)
     Case "c", "C"
       HexToDec = HexToDec + 12 * (16 ^ i)
     Case "d", "D"
       HexToDec = HexToDec + 13 * (16 ^ i)
     Case "e", "E"
       HexToDec = HexToDec + 14 * (16 ^ i)
     Case "f", "F"
       HexToDec = HexToDec + 15 * (16 ^ i)
```

```
        Case "0" To "9"
          HexToDec = HexToDec + Val(X) * 16 ^ i
        Case Else
        'MsgBox "·ÇÊ®Áù½øÖÆÊý", vbCritical, "ÐÅÏ¢ÌáÊ¾"
        HexToDec = 0
    End Select
    Next i
    End Function


    Public Function mCharToBcd(ByVal iChar As String) As Byte ' ÊäÈëµÄASCII×Ö·û
      Dim mBCD As Byte
      If iChar >= "0" And iChar <= "9" Then
        mBCD = iChar - "0"
      ElseIf iChar >= "A" And iChar <= "F" Then
        mBCD = Asc(iChar) - Asc("A") + &HA
      ElseIf iChar >= "a" And iChar <= "f" Then
        mBCD = Asc(iChar) - Asc("a") + &HA
      Else
        mBCD = &HFF
      End If
      mCharToBcd = mBCD
    End Function

    Sub mStrtoVal(str As String, ByRef strOut As arrRBuffer, strleng As Long)
      Dim i, j As Long
      Dim mLen As Long
      Dim strRev(mMAX_BUFFER_LENGTH - 1) As Byte
      mLen = strleng * 2
      j = 0
      For i = 0 To mLen - 1 Step 2
        If (mCharToBcd(Mid(str, i + 1, 1)) = &HFF Or mCharToBcd(Mid(str, i + 2, 1)) = &HFF) Then
          GoTo con
        End If
      '  strRev(j) = mCharToBcd(Mid(str, i + 1, 1)) * 16 + mCharToBcd(Mid(str, i + 2, 1))
        strRev(j) = mCharToBcd(Mid(str, i + 1, 1)) * 16 + mCharToBcd(Mid(str, i + 2, 1))
        Debug.Print Hex(strRev(j))
        j = j + 1
    con:  Next
      j = 0
      While (j < strleng)
        strOut.buf(j) = strRev(j)
        j = j + 1
      Wend
    End Sub

    Function Hex2bit(var As Byte) As String
    If var < 16 Then
      Hex2bit = "0" & Hex(var)
    Else
      Hex2bit = Hex(var)
    End If
    End Function
    Function HexToBcd(str As String) As Long              '½«ÎÄ±¾¿ò×ª»»³ÉBCDÂë
    Dim Length As Integer
    Dim X As String
    Dim i As Long
    str = Trim(str)
    Length = Len(str)
    For i = 0 To Length - 1
```

```
X = Mid(str, Length - i, 1)
Select Case X
      Case "a", "A"
        HexToBcd = HexToBcd + 10 * (16 ^ i)
      Case "b", "B"
        HexToBcd = HexToBcd + 11 * (16 ^ i)
      Case "c", "C"
        HexToBcd = HexToBcd + 12 * (16 ^ i)
      Case "d", "D"
        HexToBcd = HexToBcd + 13 * (16 ^ i)
      Case "e", "E"
        HexToBcd = HexToBcd + 14 * (16 ^ i)
      Case "f", "F"
        HexToBcd = HexToBcd + 15 * (16 ^ i)
      Case "0" To "9"
        HexToBcd = HexToBcd + Val(X) * 16 ^ i
      Case Else
      'MsgBox "·ÇÊ®Áù½øÖÆÊý", vbCritical, "ÐÅÏ¢ÌáÊ¾"
      HexToBcd = 0
End Select
Next i
End Function
Public Sub mUSBIO_NOTIFY_ROUTINE(ByVal iEventStatus As Long)
  PostMessage          frmMain.USBIO_NOTIFY_ROUTINE.hwnd,          WM_KEYUP,          iEventStatus,          0
'½«½ÓËÕµ½µÄ²â°ÎÂ¼þÖµ·¢µ½²â°Î´¦Àí³ÌÐòÖÐ
End Sub
```

Module USBIOXDLL

```
Option Explicit
' 2004.05.28, 2004.10.20, 2005.01.08, 2005.03.25, 2005.04.28
'*****************************************
'** Copyright (C) W.ch 1999-2005  **
'** Web: http://www.USB-I2C-SPI.com **
'*****************************************
'** DLL for USB interface chip USB2ISP**
'** C, VC6.0                    **
'*****************************************
'
Public Enum EEPROM_TYPE   ' EEPROMÐÍºÅ¶¨Òå
  ID_24C01 = 0
  ID_24C02 = 1
  ID_24C04 = 2
  ID_24C08 = 3
  ID_24C16 = 4
  ID_24C32 = 5
  ID_24C64 = 6
  ID_24C128 = 7
  ID_24C256 = 8
  ID_24C512 = 9
  ID_24C1024 = 10
  ID_24C2048 = 11
  ID_24C4096 = 12
End Enum

Type mUspValue
      mUspValueLow As Byte          ' 02H Öµ²ÎÊýµÍ×Ö½Ú
      mUspValueHigh As Byte         ' 03H Öµ²ÎÊý¸ß×Ö½Ú
End Type
Type mUspIndex
```

```
    mUspIndexLow As Byte            ' 04H Ë÷Òý²ÎÊýµÍ×Ö½Ú
    mUspIndexHigh  As Byte          ' 05H Ë÷Òý²ÎÊý¸ß×Ö½Ú
End Type
Type USB_SETUP_PKT                  ' USB¿ØÖÆ´«ÊäµÄ½¨Á¢½×¶ÎµÄÊý¾ÝÇëÇó°ü½á¹¹
  mUspReqType As Byte               ' 00H ÇëÇóÀàÐÍ
  mUspRequest As Byte               ' 01H ÇëÇó´úÂë
  mUspValue As mUspValue            ' 02H-03H Öµ²ÎÊý
  mUspIndex As mUspIndex            ' 04H-05H Ë÷Òý²ÎÊý
  mLength As Integer                ' 06H-07H Êý¾Ý½×¶ÎµÄÊý¾Ý³¤¶È
End Type

Public Const INVALID_HANDLE_VALUE = -1      ''ÎÎóÂë
Public Const mUSBIO_PACKET_LENGTH = 32      ' USB2ISPÖ§³ÖµÄÊý¾Ý°üµÄ³¤¶È
Public Const mUSBIO_PKT_LEN_SHORT = 8       ' USB2ISPÖ§³ÖµÄ¶ÌÊý¾Ý°üµÄ³¤¶È

Type WIN32_COMMAND                  '¶¨ÒåWIN32ÃüÁî½Ó¿Ú½á¹¹
  mFunction As Long                 'ÊâÈëËÊ±Ö¸¶¨¹¦¦ÄÜ´úÂë»òÕß¹ÜµÀºÅ
                                    'Êä³öÊ±·µ»Ø²Ù×÷×´Ì¬
  mLength As Long                   ''æÈ¡³¤¶È,·µ»Ø°óÐøÊý¾ÝµÄ³¤¶È
  mBuffer(mUSBIO_PACKET_LENGTH - 1) As Byte   'Êý¾Ý»º³åÇø,³¤¶ÈÎª0ÖÁ255B
End Type
Public mWIN32_COMMAND As WIN32_COMMAND

Public Const FILE_DEVICE_UNKNOWN = &H22
Public Const FILE_ANY_ACCESS = 0
Public Const METHOD_BUFFERED = 0
' WIN32Ó¦ÓÃ²ã½Ó¿ÚÃüÁî
Public Const IOCTL_USBIO_COMMAND = (FILE_DEVICE_UNKNOWN * (2 ^ 16) + FILE_ANY_ACCESS * 2 ^ 14 + &HF34 * 2
^ 2 + METHOD_BUFFERED)    ' ×¨ÓÃ½Ó¿Ú

Const mWIN32_COMMAND_HEAD = 8       ' WIN32ÃüÁî½Ó¿ÚµÄÍ·³¤¶È

Public Const mUSBIO_MAX_NUMBER = 16         ' ×î¶àÍ¬Ê±Á¬½ÓµÄUSB2ISPÊý

Public Const mMAX_BUFFER_LENGTH = &H1000          ' Êý¾Ý»º³åÇø×î´ó³¤¶È4096

Public Const mMAX_COMMAND_LENGTH = (mWIN32_COMMAND_HEAD + mMAX_BUFFER_LENGTH)                '
×î´óÊý¾Ý³¤¶È¼ÓÉÏÃüÁî½á¹¹Í·µÄ³¤¶È

Public Const mDEFAULT_BUFFER_LEN = &H400          ' Êý¾Ý»º³åÇøÄ¬ÈÏ³¤¶È1024

Public Const mDEFAULT_COMMAND_LEN = (mWIN32_COMMAND_HEAD + mDEFAULT_BUFFER_LEN)             '
Ä¬ÈÏÊý¾Ý³¤¶È¼ÓÉÏÃüÁî½á¹¹Í·µÄ³¤¶È



' USB2ISP¶ËµãµØÖ·
Public Const mUSBIO_ENDP_INTER_UP = &H81          ' USB2ISPµÄÖÐ¶ÏÊý¾ÝÉÏ´«¶ËµãµØÖ·
Public Const mUSBIO_ENDP_INTER_DOWN = &H1         ' USB2ISPµÄÖÐ¶ÏÊý¾ÝÏÂ´«¶ËµãµØÖ·
Public Const mUSBIO_ENDP_DATA_UP = &H82           ' USB2ISPµÄÊý¾Ý¿éÉÏ´«¶ËµãµØÖ·
Public Const mUSBIO_ENDP_DATA_DOWN = &H2          ' USB2ISPµÄÊý¾Ý¿éÏÂ´«¶ËµãµØÖ·

' Éè±¸²ã½Ó¿Ú¼°µÄ¹ÜµÀ²Ù×÷ÃüÁî
Public Const mPipeDeviceCtrl = &H4          ' USB2ISPµÄ×Ö¸ï¿ØÖÆ¹ÜµÀ
Public Const mPipeInterUp = &H5             ' USB2ISPµÄÖÐ¶ÏÊý¾ÝÉÏ´«¹ÜµÀ
Public Const mPipeDataUp = &H6              ' USB2ISPµÄÊý¾Ý¿éÉÏ´«¹ÜµÀ
Public Const mPipeDataDown = &H7            ' USB2ISPµÄÊý¾Ý¿éÏÂ´«¹ÜµÀ
```

```
' Ó¦ÓÃ²ã½Ó¿ÚµÄ¹¦ÄÜ´úÂë
Public Const mFuncNoOperation = &H0              ' ÎÞ²Ù×÷
Public Const mFuncGetVersion = &H1               ' »ñÈ¡Çý¶¯³ÌÐò°æ±¾ºÅ
Public Const mFuncGetConfig = &H2               ' »ñÈ¡USBÉè±¸ÅäÖÃÃèÊö·û
Public Const mFuncSetTimeout = &H9               ' ÉèÖÃUSBÍ¨Ñ¶³¬Ê±
Public Const mFuncSetExclusive = &HB             ' ÉèÖÃ¶ÀÕ¼Ê¹ÓÃ
Public Const mFuncResetDevice = &HC              ' ¸´Î»USBÉè±¸
Public Const mFuncResetPipe = &HD                ' ¸´Î»USB¹ÜµÀ
Public Const mFuncAbortPipe = &HE                ' È¡ÏûUSB¹ÜµÀµÄÊý¾ÝÇëÇó

' USB2ISP²¢¿Ú×¨ÓÃµÄ¹¦ÄÜ´úÂë
Public Const mFuncSetParaMode = &HF              ' ÉèÖÃ²¢¿ÚÄ£Ê½
Public Const mFuncReadData0 = &H10               ' ´Ó²¢¿Ú¶ÁÈ¡ÊýÝ¿é0
Public Const mFuncReadData1 = &H11               ' ´Ó²¢¿Ú¶ÁÈ¡ÊýÝ¿é1
Public Const mFuncWriteData0 = &H12              ' Ïò²¢¿ÚÐ´ÈëÊýÝ¿é0
Public Const mFuncWriteData1 = &H13              ' Ïò²¢¿ÚÐ´ÈëÊýÝ¿é1
Public Const mFuncWriteRead = &H14               ' ÏÈÊä³öÔÙÊäÈë


' USBÉè±¸±ê×¼ÇëÇó´úÂë
Public Const mUSB_CLR_FEATURE = &H1
Public Const mUSB_SET_FEATURE = &H3
Public Const mUSB_GET_STATUS = &H0
Public Const mUSB_SET_ADDRESS = &H5
Public Const mUSB_GET_DESCR = &H6
Public Const mUSB_SET_DESCR = &H7
Public Const mUSB_GET_CONFIG = &H8
Public Const mUSB_SET_CONFIG = &H9
Public Const mUSB_GET_INTERF = &HA
Public Const mUSB_SET_INTERF = &HB
Public Const mUSB_SYNC_FRAME = &HC

' USB2ISP¿ØÖÆ´«ÊäµÄ³§Éí×¨ÓÃÇëÇóÅàÐÍ
Public Const mUSBIO_VENDOR_READ = &HC0           ' Í¨¹ý¿ØÖÆ´«ÊäÊµÏÖµÄUSB2ISP³§Éí×¨ÓÃ¶Á²Ù×÷
Public Const mUSBIO_VENDOR_WRITE = &H40          ' Í¨¹ý¿ØÖÆ´«ÊäÊµÏÖµÄUSB2ISP³§Éí×¨ÓÃÐ´²Ù×÷

' USB2ISP¿ØÖÆ´«ÊäµÄ³§Éí×¨ÓÃÇëÇó´úÂë
Public Const mUSBIO_PARA_INIT = &HB1             ' ³õÊ¼»¯²¢¿Ú
Public Const mUSBIO_I2C_STATUS = &H52            ' »ñÈ¡I2C½Ó¿ÚµÄ×´Ì¬
Public Const mUSBIO_I2C_COMMAND = &H53           ' ·¢³öI2C½Ó¿ÚµÄÃüÁî


Public Const mUSBIOA_CMD_I2C_STM_STA = &H74      ' I2C½Ó¿ÚµÄÃüÁîÁî÷:²úÉúÆðÊ¼Î»
Public Const mUSBIOA_CMD_I2C_STM_STO = &H75      ' I2C½Ó¿ÚµÄÃüÁîÁî÷:²úÉúÍ£Ö¹Î»
Public Const mUSBIOA_CMD_I2C_STM_OUT = &H0       'I2C½Ó¿ÚµÄÃüÁîÁî÷:Êä³öÊý¾ÝÎ»5-Î»0Î³¤¶È,°óÐø×Ö½ÚÎªÊý¾Ý,0³¤¶ÈÔòÖ·¢ËÍÒ»¸ö×Ö½Ú²¢·µ»ØÓ¦´ð
Public Const mUSBIOA_CMD_I2C_STM_IN = &HC0       ' I2C½Ó¿ÚµÄÃüÁîÁî÷:ÊäÈëÊý¾ÝÎ»5-Î»0Î³¤¶È,0³¤¶ÈÔòÖ»½ÓÊÕÖ»¸ö×Ö½Ú²¢·¢ËÍÎÞÓ¦´ð
Public Const mUSBIOA_CMD_I2C_STM_SET = &H60      '
I2C½Ó¿ÚµÄÃüÁîÁî÷:ÉèÖÃ²ÎÊý,Î»2=SPIµÄÌ/OÊý(0=µ¥Èëµ¥³ö,1=Ë«ÈëË«³ö),Î»1Î»0=I2CËÙ¶È(00=µÍËÙ,01=±ê×¼,10=¿ìËÙ,11=¸ßËÙ)
Public Const mUSBIOA_CMD_I2C_STM_US = &H40       ' I2C½Ó¿ÚµÄÃüÁîÁî÷:ÒÔÎ¢ÃëÎªµ¥Î»ÑÓÊ±,Î»3-Î»0Î³ÑÓÊ±Öµ
Public Const mUSBIOA_CMD_I2C_STM_MS = &H50       ' I2C½Ó¿ÚµÄÃüÁîÁî÷:ÒÔÙñÃëÎªµ¥Î»ÑÓÊ±,Î»3-Î»0Î³ÑÓÊ±Öµ
Public Const mUSBIOA_CMD_I2C_STM_DLY = &HF       ' I2C½Ó¿ÚµÄÃüÁîÁî÷µ¥¸öÃüÁîÑÓÊ±µÄ×î´óÖµ
Public Const mUSBIOA_CMD_I2C_STM_END = &H0       ' I2C½Ó¿ÚµÄÃüÁîÁî÷:ÃüÁî°üÌáÇ°½áÊø


' Ö±½ÓÊäÈëµÄ×´Ì¬ÐÅºÅµÄÎ»¶¨Òå
```

```
Public Const mStateBitERR = &H100              ' Ö»¶Á,ERR#Òý½ÂÊäËë×´Ì¬,1:¸ßµçÆ½,0:µÍµçÆ½
Public Const mStateBitPEMP = &H200             ' Ö»¶Á,PEMPÒý½ÂÊäËë×´Ì¬,1:¸ßµçÆ½,0:µÍµçÆ½
Public Const mStateBitINT = &H400              ' Õ»¶Á,INT#Òý½ÂÊäËë×´Ì¬,1:¸ßµçÆ½,0:µÍµçÆ½
Public Const mStateBitSLCT = &H800             ' Ö»¶Á,SLCTÒý½ÂÊäËë×´Ì¬,1:¸ßµçÆ½,0:µÍµçÆ½
Public Const mStateBitSDA = &H800000           ' Ö»¶Á,SDAÒý½ÂÊäËë×´Ì¬,1:¸ßµçÆ½,0:µÍµçÆ½



Declare Function USBIO_OpenDevice Lib "USBIOX.DLL" (ByVal iIndex As Long) As Long
' ´ò¿ªUSB2ISPÉè±¸,·µ»Ø¾ä±ú,³ö´íÔò¦ÞÐ§
' iIndex  Ö¸¶¨USB2ISPÉè±¸Ðò°Å,0¶ÔÓ¦µÚÒ»¸öÉè±¸

Declare Sub USBIO_CloseDevice Lib "USBIOX.DLL" (ByVal iIndex As Long)
' ¹Ø±ÕUSB2ISPÉè±¸
' iIndex    Ö¸¶¨USB2ISPÉè±¸Ðò°Å

Declare Function USBIO_GetVersion Lib "USBIOX.DLL" () As Long

' »ñµÃDLL°æ±¾°Å,·µ»Ø°æ±¾°Å

Declare Function USBIO_DriverCommand Lib "USBIOX.DLL" (ByVal iIndex As Long, ByRef ioCommand As
WIN32_COMMAND) As Long
' Ö±½Ó´«µÝÃüÁî,øÇý¶¯³ÌÐò,³ö´íÔò·µ»Ø0,·ñÔò·µ»ØÊý¾Ý³¤¶È
' iIndex,  ' Ö¸¶¨USB2ISPÉè±¸Ðò°Å,V1.6ÒÔÉÏDLLÒ²¿ÉÒÔÊÇÉè±¸´ò¿ªóµÄ¾ä±ú
' ioCommand  ÃüÁî½á¹¹µÄµØÖ·
' ¸Ã³ÌÐòÔÚµ÷ÓÃ°ó·µ»ØÊý¾Ý³¤¶È,²¢ÇÒÈÔÊ»·µ»ØÃüÁî½á¹¹,Èç¹ûÊÇ¶Á²Ù×÷,ÔòÊý¾Ý·µ»ØÔÚÃüÁî½á¹¹ÖÐ,
'
·µ»ØµÄÊý¾Ý³¤¶ÈÔÚ²Ù×÷Ê§°ÜÊ±Î³Ö0,²Ù×÷³É¹¦Ê±Î³ÖÕû¸öÃüÁî½á¹¹µÄ³¤¶È,ÀýÈç¶ÁÒ»¸ö×Ö½Ú,Ôò·µ»ØmWIN32_COMMAND_H
EAD+1,
' ÃüÁî½á¹¹ÔÚµ÷ÓÃÇ°,·Ö±ðÌá¹©:¹ÜµÀºÅ»òÕßÃüÁî¦ÄÜ´úÂë,´æÈ¡Êý¾ÝµÄ³¤¶È(¿ÉÑ¡),Êý¾Ý(¿ÉÑ¡)
' ÃüÁî½á¹¹ÔÚµ÷ÓÃºó,·Ö±ð·µ»Ø:²Ù×÷×´Ì¬´úÂë,ºóÐøÊý¾ÝµÄ³¤¶È(¿ÉÑ¡),
'  ²Ù×÷×´Ì¬´úÂëÊÇÓÉWINDOWS¶¨Òåµ´úÂë,¿ÉÒÔ²Î¿¼NTSTATUS.H,
'  ºóÐøÊý¾ÝµÄ³¤¶ÈÊÇÖ,¶Á²Ù×÷·µ»ØµÄÊý¾Ý³¤¶È,Êý¾Ý´æ·ÅÔÚËæºóµÄ»º³åÇøÖÐ,¶ÔÓÚÐ´²Ù×÷Ò»°ãÎª0

Declare Function USBIO_GetDrvVersion Lib "USBIOX.DLL" () As Long

' »ñµÃÇý¶¯³ÌÐò°æ±¾°Å,·µ»Ø°æ±¾°Å,³ö´íÔò·µ»Ø0

Declare Function USBIO_ResetDevice Lib "USBIOX.DLL" (ByVal iIndex As Long) As Boolean

'¸´Î»USBÉè±¸
' iIndex  Ö¸¶¨USB2ISPÉè±¸Ðò°Å

Declare Function USBIO_GetDeviceDescr Lib "USBIOX.DLL" (ByVal iIndex As Long, ByRef oBuffer As Any, ByRef ioLength As
Long) As Boolean
' ¶ÁÈ¡Éè±¸ÃèÊö·û
' iIndex,  Ö¸¶¨USB2ISPÉè±¸Ðò°Å
' oBuffer  Ö¸ÏòÒ»¸ö×ã¹»´óµÄ»º³åÇø,ÓÃÓÚ±£´æÃèÊö·û
' ioLength  Ö¸Ïò³¤¶ÈµÄÔª,ÊäÈëÊ±Î³¤¶È¡µÄÈ¡µÃ¶ÁÈ¡µÄ³¤¶È

Declare Function USBIO_GetConfigDescr Lib "USBIOX.DLL" (ByVal iIndex As Long, ByRef oBuffer As Any, ByRef ioLength As
Long) As Boolean
' ¶ÁÈ¡ÅäÖÃÃèÊö·û
' iIndex,  Ö¸¶¨USB2ISPÉè±¸Ðò°Å
' oBuffer,  Ö¸ÏòÒ»¸ö×ã¹»´óµÄ»º³åÇø,ÓÃÓÚ±£´æÃèÊö·û
' ioLength  Ö¸Ïò³¤¶ÈµÄÔª,ÊäÈëÊ±Î³¤¶È¡µÄÈ¡µÃ¶ÁÈ¡µÄ³¤¶È
```

```
Declare Function USBIO_SetIntRoutine Lib "USBIOX.DLL" (ByVal iIndex As Long, ByVal iIntRoutine As Long) As Boolean
'Éè¶¨ÖÐ¶Ï·þÎñ³ÌÐò
' Ö¸¶¨USB2ISPÉè±¸Ðò°Å
'Ö¸¶¨ÖÐ¶Ï·þÎñ³ÌÐò,Î ªNULLÔòÈ¡ÏûÖÐ¶Ï·þÎñ,·ñÔòÔÚÖÐ¶ÏÊ±µ÷ÓÃ¸Ã³ÌÐò

Declare Function USBIO_ReadInter Lib "USBIOX.DLL" (ByVal iIndex As Long, ByRef iStatus As Long) As Boolean
' ¶ÁÈ¡ÖÐ¶ÏÊý¾Ý
' iIndex, Ö¸¶¨USB2ISPÉè±¸Ðò°Å
' iStatus Ö¸ÏòÒ»¸öÊ«×Öµ¥Ô ª,ÓÃÓÚ±£´æ¶ÁÈ¡µÄÖÐ¶Ï×´Ì¬Êý¾Ý,¼ûÏÂÐÐ
' Î»7-Î»0¶ÔÓ¦USB2ISPµÄD7-D0Òý½Å
'  Î»8¶ÔÓ¦USB2ISPµÄERR#Òý½Å,  Î»9¶ÔÓ¦USB2ISPµÄPEMPÒý½Å,  Î»10¶ÔÓ¦USB2ISPµÄINT#Òý½Å,
Î»11¶ÔÓ¦USB2ISPµÄSLCTÒý½Å

Declare Function USBIO_AbortInter Lib "USBIOX.DLL" (ByVal iIndex As Long) As Boolean
' ·ÅÆúÖÐ¶ÏÊý¾Ý¶Á²Ù×÷
' iIndex Ö¸¶¨USB2ISPÉè±¸Ðò°Å

Declare Function USBIO_ReadData0 Lib "USBIOX.DLL" (ByVal iIndex As Long, ByRef oBuffer As Any, ByRef ioLength As
Long) As Boolean
' ´Ó0#¶Ë¿Ú¶ÁÈ¡ÊÝ¾Ý¿é
' iIndex Ö¸¶¨USB2ISPÉè±¸Ðò°Å
' oBuffer Ö¸ÏòÒ»¸ö×ã¹»´óµÄ»º³åÇø,ÓÃÓÚ±£´æ¶ÁÈ¡µÄÊý¾Ý
' ioLength Ö¸Ïò³¤¶È¼¥Ô ª,ÊäËëÊ±Î ª×¼±¸¶ÁÈ¡µÄ³¤¶È,·µ»Ø°óÎ ªÊµ¼Ê¶ÁÈ¡µÄ³¤¶È

Declare Function USBIO_ReadData1 Lib "USBIOX.DLL" (ByVal iIndex As Long, ByRef oBuffer As Any, ByRef ioLength As
Long) As Boolean
' ´Ó1#¶Ë¿Ú¶ÁÈ¡ÊÝ¾Ý¿é
' iIndex, Ö¸¶¨USB2ISPÉè±¸Ðò°Å
' oBuffer Ö¸ÏòÒ»¸ö×ã¹»´óµÄ»º³åÇø,ÓÃÓÚ±£´æ¶ÁÈ¡µÄÊý¾Ý
' ioLength Ö¸Ïò³¤¶È¼¥Ô ª,ÊäËëÊ±Î ª×¼±¸¶ÁÈ¡µÄ³¤¶È,·µ»Ø°óÎ ªÊµ¼Ê¶ÁÈ¡µÄ³¤¶È

Declare Function USBIO_AbortRead Lib "USBIOX.DLL" (ByVal iIndex As Long) As Boolean
' ·ÅÆúÊý¾Ý¿é¶Á²Ù×÷
' iIndex Ö¸¶¨USB2ISPÉè±¸Ðò°Å

Declare Function USBIO_WriteData0 Lib "USBIOX.DLL" (ByVal iIndex As Long, ByRef iBuffer As Any, ByRef ioLength As Long)
As Boolean
' Ïò0#¶Ë¿ÚÐ´³öÊý¾Ý¿é
' iIndex, Ö¸¶¨USB2ISPÉè±¸Ðò°Å
' iBuffer Ö¸ÏòÒ»¸ö³Çø,·ÅÖÃ×¼±¸Ð´³öµÄÊý¾Ý
' ioLength Ö¸Ïò³¤¶È¼¥Ô ª,ÊäËëÊ±Î ª×¼±¸Ð´³öµÄ³¤¶È,·µ»Ø°óÎ ªÊµ¼ÊÐ´³öµÄ³¤¶È

Declare Function USBIO_WriteData1 Lib "USBIOX.DLL" (ByVal iIndex As Long, ByRef iBuffer As Any, ByRef ioLength As Long)
As Boolean
' Ïò1#¶Ë¿ÚÐ´³öÊý¾Ý¿é
' iIndex, Ö¸¶¨USB2ISPÉè±¸Ðò°Å
' iBuffer, Ö¸ÏòÒ»¸ö³Çø,·ÅÖÃ×¼±¸Ð´³öµÄÊý¾Ý
' ioLength Ö¸Ïò³¤¶È¼¥Ô ª,ÊäËëÊ±Î ª×¼±¸Ð´³öµÄ³¤¶È,·µ»Ø°óÎ ªÊµ¼ÊÐ´³öµÄ³¤¶È

Declare Function USBIO_AbortWrite Lib "USBIOX.DLL" (ByVal iIndex As Long) As Boolean
' ·ÅÆúÊý¾Ý¿éÐ´²Ù×÷
' iIndex Ö¸¶¨USB2ISPÉè±¸Ðò°Å

Declare Function USBIO_GetStatus Lib "USBIOX.DLL" (ByVal iIndex As Long, ByRef iStatus As Long) As Boolean
' Í¨ýUSB2ISPÖ±½ÓÊäËëÊý¾Ý°í×´Ì¬
' iIndex, Ö¸¶¨USB2ISPÉè±¸Ðò°Å
```

```
' iStatus Ö¸ÏòÒ»¸öË«×ÔµÝÔª,ÓÃÓÚ±£´æ×´Ì¬ÊýÝ,¼ûÏÂÐÐ
' Î»7-Î»0¶ÔÓ¦USB2ISPµÄD7-D0ÒýÅ
'          Î»8¶ÔÓ¦USB2ISPµÄERR#ÒýÅ,     Î»9¶ÔÓ¦USB2ISPµÄPEMPÒýÅ,     Î»10¶ÔÓ¦USB2ISPµÄINT#ÒýÅ,
Î»11¶ÔÓ¦USB2ISPµÄSLCTÒýÅ, Î»23¶ÔÓ¦USB2ISPµÄSDAÒýÅ
'                                                                    Î»13¶ÔÓ¦USB2ISPµÄBUSY/WAIT#ÒýÅ,
Î»14¶ÔÓ¦USB2ISPµÄAUTOFD#/DATAS#ÒýÅ,Î»15¶ÔÓ¦USB2ISPµÄSLCTIN#/ADDRS#ÒýÅ

Declare Function USBIO_ReadI2C Lib "USBIOX.DLL" (ByVal iIndex As Long, ByVal iDevice As Byte, ByVal iAddr As Byte,
ByRef oByte As Byte) As Boolean

' ´ÓI2C½Ó¿Ú¶ÁÈ¡Ò»¸ö×Ö½ÚÊý¾Ý
' iIndex,  Ö¸¶¨USB2ISPÉè±¸Đò°Å
' iDevice,  µÍ7Î»Ö¸¶¨I2CÉè±¸µØÖ·
' iAddr,  Ö¸¶¨Êý¾Ýµ¥Ôªµ¥ÄµØÖ·
' oByte  Ö¸ÏòÒ»¸ö×Ö½ÚµÝÔª,ÓÃÓÚ±£´æ¶ÁÈ¡µÄ×Ö½ÚÊý¾Ý

Declare Function USBIO_WriteI2C Lib "USBIOX.DLL" (ByVal iIndex As Long, ByVal iDevice As Byte, ByVal iAddr As Byte,
ByVal iByte As Byte) As Boolean

' ÏòI2C½Ó¿ÚĐ´ÈëÒ»¸ö×Ö½ÚÊý¾Ý
' iIndex,  Ö¸¶¨USB2ISPÉè±¸Đò°Å
' iDevice,  µÍ7Î»Ö¸¶¨I2CÉè±¸µØÖ·
' iAddr,  Ö¸¶¨Êý¾Ýµ¥Ôªµ¥ÄµØÖ·
' iByte  ´ýĐ´ÈëµÄ×Ö½ÚÊý¾Ý

Declare Function USBIO_SetExclusive Lib "USBIOX.DLL" (ByVal iIndex As Long, ByVal iExclusive As Long) As Boolean
' ÉèÖÃ¶ÀÕ¼Ê¹ÓÃµ±Ç°USB2ISPÉè±¸
' iIndex,  Ö¸¶¨USB2ISPÉè±¸Đò°Å
' iExclusive  Î²0ÔòÉè±¸¿ÉÒÔ¹²ïÊ¹ÓÃ,·Ç0ÔòÔ¶ÀÕ¼Ê¹ÓÃ

Declare Function USBIO_SetTimeout Lib "USBIOX.DLL" (ByVal iIndex As Long, ByVal iWriteTimeout As Long, ByVal
iReadTimeout As Long) As Boolean
' ÉèÖÃUSBÊý¾Ý¶ÁĐ´µÄ³¬Ê±
' iIndex,  // Ö¸¶¨USB2ISPÉè±¸Đò°Å
' iWriteTimeout  Ö¸¶¨USBĐ´³öÊý¾Ý¿éµÄ³¬Ê±Ê±¼ä,ÒÔºÁÃëmSÎª¼Î»,0xFFFFFFFFÖ¸¶¨²»³¬Ê±(Ä¬ÈÏÖµ)
' iReadTimeout  Ö¸¶¨USB¶ÁÈ¡Êý¾Ý¿éµÄ³¬Ê±Ê±¼ä,ÒÔºÁÃëmSÎª¼Î»,0xFFFFFFFFÖ¸¶¨²»³¬Ê±(Ä¬ÈÏÖµ)

Declare Function USBIO_ReadData Lib "USBIOX.DLL" (ByVal iIndex As Long, ByRef oBuffer As Any, ByRef ioLength As Long)
As Boolean
' ¶ÁÈ¡Êý¾Ý¿é
' iIndex,  Ö¸¶¨USB2ISPÉè±¸Đò°Å
' oBuffer,  Ö¸ÏòÒ»¸ö´óµÄ»º³åÇø,ÓÃÓÚ±£´æ¶ÁÈ¡µÄÊý¾Ý
' ioLength  Ö¸Ïò³¤¶È¼ÄÔª,ÊäËëÊ±Î²×¼±¸¶ÁÈ¡µÄ³¤¶È,·µ»Ø°óÎªÊµ¼ÊÊ¶ÁÈ¡µÄ³¤¶È

Declare Function USBIO_WriteData Lib "USBIOX.DLL" (ByVal iIndex As Long, ByRef iBuffer As Any, ByRef ioLength As Long)
As Boolean
' Đ´öÊý¾Ý¿é
' iIndex,  Ö¸¶¨USB2ISPÉè±¸Đò°Å
' iBuffer,  Ó¸ÏòÒ»¸ö»º³åÇø,·ÅÖÃ×¼±¸Đ´³öµÄÊý¾Ý
' ioLength  Ö¸Ïò³¤¶È¼ÄÔª,ÊäËëÊ±Î²×¼±¸Đ´³öµÄ³¤¶È,·µ»Ø°óÎªÊµ¼ÊĐ´³öµÄ³¤¶È

Declare Function USBIO_GetDeviceName Lib "USBIOX.DLL" (ByVal iIndex As Long) As Long          ""
' ·µ»ØÖ¸ÏòUSB2ISPÉè±¸Ãû³ÆµÄ»º³åÇø,³ö´íÔò·µ»ØNULL
' iIndex  Ö¸¶¨USB2ISPÉè±¸Đò°Å,0¶ÔÓ¦µÚÒ»¸öÉè±¸

Declare Function USBIO_FlushBuffer Lib "USBIOX.DLL" (ByVal iIndex As Long) As Boolean
' Çå¿ÕUSB2ISPµÄ»º³åÇø
' iIndex  Ö¸¶¨USB2ISPÉè±¸Đò°Å
```

```
Declare Function USBIO_WriteRead Lib "USBIOX.DLL" (ByVal iIndex As Long, ByVal iWriteLength As Long, ByRef iWriteBuffer
As Any, ByVal iReadStep As Long, ByVal iReadTimes As Long, ByRef oReadLength As Long, ByRef oReadBuffer As Any) As
Boolean
' USBIO_WriteRead  Ö´ÐÐÊý¾ÝÁ÷ÃüÁî,ÏÈÊä³öÔÙÊäËë
' iIndex,   Ö¸¶¨USB2ISPÉè±¸Đò°Å
' iWriteLength,  Đ´³¤¶È,×¼±¸Đ´³öµÄ³¤¶È
' iWriteBuffer,   Ö¸ÏòÒ»¸ö»º³åÇø,·ÅÖÃ×¼±¸Đ´³öµÄÊý¾Ý
' iReadStep,   ×¼±¸¶ÁÈ¡µÄµ¥¸ö¿éµÄ³¤¶È, ×¼±¸¶ÁÈ¡µÄ×Ü³¤¶ÈÎª(iReadStep*iReadTimes)
' iReadTimes,   ×¼±¸¶ÁÈ¡µÄ´ÎÊý
' oReadLength,   Ö¸Ïò³¤¶Èµ¥Ôª,·µ»Ø°óÎª¼Ê¶ÁÈ¡µÄ³¤¶È
' oReadBuffer    Ö¸ÏòÒ»¸ö×ã¹»´óµÄ»º³åÇø,ÓÃÓÚ±£´æ¶ÁÈ¡µÄÊý¾Ý

Declare Function USBIO_SetStream Lib "USBIOX.DLL" (ByVal iIndex As Long, ByVal iMode As Long) As Boolean
' USBIO_SetStream  ÉèÖÃ´®¿ÚÁ÷Ä£Ê½
' iIndex,   Ö¸¶¨USB2ISPÉè±¸Đò°Å
' iMode    Ö¸¶¨Ä£Ê½,¼´öÏÂÐÐ
' Î»1-Î»0: I2C½Ó¿ÚËÙ¶È/SCLÆµÂÊ, 00=µÍËÙ/20KHz,01=±ê×¼/100KHz,10=¿ìËÙ/400KHz,11=,ßËÙ/750KHz
' Î»2:   SPIµÄI/OÊý/IOÒý½Å, 0=µ¥Èëµ¥ö(D5³ö/D7Èë),1=Ë«ËëË«³ö(D5³öD4³ö/D7ËëD6Èë)
' Î»7:   SPI×Ö½ÚÖĐµÄÎ»Ë³Đò, 0=µÍÎ»ÔÚÇ°, 1=¸ßÎ»ÔÚÇ°
' ÆäËü±£Áô,±ØĐèÎª0

Declare Function USBIO_SetDelaymS Lib "USBIOX.DLL" (ByVal iIndex As Long, ByVal iDelay As Long) As Boolean
' USBIO_SetDelaymS   ÉèÖÃÓ²¼þÒì²½ÑÓÊ±,µ÷ÓÃº óÜ¿ì·µ»Ø,¶øÔÚÎÂÒ»,öÁ÷²Ù×÷Ö®Ç°ÑÓÊ±Ö¸¶¨°ÁÃëÊý
' iIndex,   Ö¸¶¨USB2ISPÉè±¸Đò°Å
' iDelay    Ö¸¶¨ÑÓÊ±µÄ°ÁÃëÊý

Declare Function USBIO_StreamI2C Lib "USBIOX.DLL" (ByVal iIndex As Long, ByVal iWriteLength As Long, ByRef iWriteBuffer
As Any, ByVal iReadLength As Long, ByRef oReadBuffer As Any) As Boolean
' USBIO_StreamI2C   ´¦ÀíI2CÊý¾ÝÁ÷,2Îß½Ó¿Ú,Ê±ÔÓÏßÎªSCLÒý½Å,Êý¾ÝÏßÎªSDAÒý½Å(×¼Ë«ÏòI/O),ËÙ¶ÈÔ¼56K×Ö½Ú
' iIndex,   Ö¸¶¨USB2ISPÉè±¸Đò°Å
' iWriteLength,   ×¼±¸Đ´³öµÄÊý¾Ý×Ö½ÚÊý
' iWriteBuffer,   Ö¸ÏòÒ»¸ö»º³åÇø,·ÅÖÃ×¼±¸Đ´³öµÄÊý¾Ý,Ê××Ö½ÚÍ¨³£ÊÇI2CÉè±¸µØÖ·¼°¶ÁĐ´·½ÏòÎ»
' iReadLength,   ×¼±¸¶ÁÈ¡µÄÊý¾Ý×Ö½ÚÊý
' oReadBuffer   Ö¸ÏòÒ»¸ö»º³åÇø,·µ»Ø°óÊÇ¶ÁÈëµÄÊý¾Ý
Declare Function USBIO_ReadEEPROM Lib "USBIOX.DLL" (ByVal iIndexas As Long, ByVal iEepromID As EEPROM_TYPE,
ByVal iAddr As Long, ByVal iLength As Long, ByRef oBuffer As Any) As Boolean
' i   Index Ö¸¶¨USB2ISPÉè±¸Đò°Å
' iEepromID  Ö¸¶¨EEPROMĐÍºÅ
' iAddr     Ö¸¶¨Êý¾Ý¿ªÊ¼µØÖ·
' iLength    ×¼±¸¶ÁÈ¡µÄÊý¾Ý×Ö½ÚÊý
' oBuffer    Ö¸ÏòÒ»¸ö»º³åÇø,·µ»Ø°óÊÇ¶ÁÈëµÄÊý¾Ý

Declare Function USBIO_WriteEEPROM Lib "USBIOX.DLL" (ByVal iIndex As Long, ByVal iEepromID As EEPROM_TYPE,
ByVal iAddr As Long, ByVal iLength As Long, ByRef iBuffer As Any) As Boolean
' iIndex,   Ö¸¶¨USB2ISPÉè±¸Đò°Å
' iEepromID, Ö¸¶¨EEPROMĐÍºÅ
' iAddr,    Ö¸¶¨Êý¾Ý¿ªÊ¼µØÖ·
' iLength,   ×¼±¸Đ´³öµÄÊý¾Ý×Ö½ÚÊý
' iBuffer   Ö¸ÏòÒ»¸ö»º³åÇø,·ÅÖÃ×¼±¸Đ´³öµÄÊý¾Ý

Declare Function USBIO_SetBufUpload Lib "USBIOX.DLL" (ByVal iIndex As Long, ByVal iEnableOrClear As Long) As Boolean
' Î²0Ôò½ûÖ¹ÄÚ²¿»º³åÉÏ´«Ä£Ê½,Ê¹ÓÃÖ±½ÓÉÏ´«,·Ç0ÔòÆôÓÃÄÚ²¿»º³åÉÏ´«Ä£Ê½²¢Çå³ý»º³åÇøÖĐµÄÄÚÑÓĐÊý¾Ý
' USBIO_SetBufUpload  ' Éè¶¨ÄÚ²¿»º³åÉÏ´«Ä£Ê½
' iIndex,        0Ö¸¶¨USB2ISPÉè±¸Đò°Å,0¶ÔÓ¦µÚÖ»¸öÉè±¸
'                                                                                    iEnableOrClear
Î²0Ôò½ûÖ¹ÄÚ²¿»º³åÉÏ´«Ä£Ê½,Ê¹ÓÃÖ±½ÓÉÏ´«,·Ç0ÔòÆôÓÃÄÚ²¿»º³åÉÏ´«Ä£Ê½²¢Çå³ý»º³åÇøÖĐµÄÄÚÑÓĐÊý¾Ý
'
Èç¹ûÆôÓÃÄÚ²¿»º³åÉÏ´«Ä£Ê½,ÄÇÃ´USB2ISPÇý¶¯³ÌÐò´´½¨Ïß³Ì×Ô¶¯½ÓÊÕUSBÉÏ´«Êý¾Ý¼´½ÄÚ²¿»º³åÇø,Í¬Ê±Çå³ý»º³åÇøÖĐ
µÄÄÚÑÓĐÊý¾Ý,µ±Ó¦ÓÃ³ÌÐòµ÷ÓÃUSBIO_ReadData°ó¶¨Á¢¼´·µ»Ø»º³åÇøÖĐµÄÄÚÑÓĐÊý¾Ý
```

```
Declare Function USBIO_QueryBufUpload Lib "USBIOX.DLL" (ByVal iIndex As Long) As Long
' USBIO_QueryBufUpload  ²éÑ¯ÄÚ²¿¿ÉÏ´«»º³âÇøÖÐµÄÒÑÓÐÊý¾Ý°ü¸öÊý,³É¹¦·µ»ØÊý¾Ý°ü¸öÊý,³ö´í·µ»Ø-1
' iIndex              Ö¸¶¨USB2ISPÉè±¸Đò°Å,0¶ÔÓ¦µÚÒ»¸öÉè±¸

Declare Function USBIO_SetBufDownload Lib "USBIOX.DLL" (ByVal iIndex As Long, ByVal iEnableOrClear As Long) As
Boolean
' USBIO_SetBufDownload  Éè¶¨ÄÚ²¿»º³åÏÂ´«Ä£Ê½
' iIndex,              Ö¸¶¨USB2ISPÉè±¸Đò°Å,0¶ÔÓ¦µÚÒ»¸öÉè±¸,
'                                                                                    iEnableOrClear
Î³0Ôò½ûÖ¹ÄÚ²¿»º³åÏÂ´«Ä£Ê½,Ê¹ÓÃÖ±½ÓÏÂ´«,·Ç0Ôò/ÆôÓÃÄÚ²¿»º³åÏÂ´«Ä£Ê½²¢Çå³ý»º³åÇøÖÐµÄÒÑÓÐÊý¾Ý
'
Èç¹ûÆôÓÃÄÚ²¿»º³åÏÂ´«Ä£Ê½,ÄÇÃ´µ±Ó¦ÓÃ³ÌÐòµ÷ÓÃUSBIO_WriteData°ó½«½ö½öÊÇ½«USBÏÂ´«Êý¾Ý·Åµ½ÄÚ²¿»º³åÇø²¢Á
¢¼´·µ»Ø,¶øÓÉUSB2ISPÇý¶¯³ÌÐò´´½¨µÄÏß³Ì×Ô¶¯·¢ËÍÖ±µ½íê±Ï

Declare Function USBIO_QueryBufDownload Lib "USBIOX.DLL" (ByVal iIndex As Long) As Long                    '
Ö¸¶¨USB2ISPÉè±¸Đò°Å,0¶ÔÓ¦µÚÒ»¸öÉè±¸,
' USBIO_QueryBufDownload  ²éÑ¯ÄÚ²¿¿ÏÂ´«»º³åÇøÖÐµÄÊ£ÓàÊý¾Ý°ü¸öÊý(ÉĐÎ´·¢ËÍ),³É¹¦·µ»ØÊý¾Ý°ü¸öÊý,³ö´í·µ»Ø-1
' iIndex              Ö¸¶¨USB2ISPÉè±¸Đò°Å,0¶ÔÓ¦µÚÒ»¸öÉè±¸,


Declare Function USBIO_ResetInter Lib "USBIOX.DLL" (ByVal iIndex As Long) As Boolean
' USBIO_ResetInter  ¸´Î»ÖĐ¶ÏÊý¾Ý¶Á²Ù×÷
' iIndex          Ö¸¶¨USB2ISPÉè±¸Đò°Å

Declare Function USBIO_ResetRead Lib "USBIOX.DLL" (ByVal iIndex As Long) As Boolean
' USBIO_ResetRead  ¸´Î»Êý¾Ý¿é¶Á²Ù×÷
' iIndex         Ö¸¶¨USB2ISPÉè±¸Đò°Å

Declare Function USBIO_ResetWrite Lib "USBIOX.DLL" (ByVal iIndex As Long) As Boolean
'   USBIO_ResetRead  ¸´Î»Êý¾Ý¿é¶Á²Ù×÷
'   iIndex          Ö¸¶¨USB2ISPÉè±¸Đò°Å

'typedef    VOID    ( CALLBACK  * mUSBIO_NOTIFY_ROUTINE ) (  ' Éè±¸ÊÂ¼þÏ¨Ö¤»Øµ÷³ÌĐò
'   ULONG          iEventStatus );  ' Éè±¸ÊÂ¼þº¯Íµ±Ç°×´Ì¬(ÔÚÏÂĐĐ¶¨Òå): 0=Éè±¸,°Î³öÊÂ¼þ, 3=Éè±¸,²åÈëÊÂ¼þ

Public Const USBIO_DEVICE_ARRIVAL = 3              ' Éè±¸,²åÈëÊÂ¼þ,ÒÑ¾²åÈë
Public Const USBIO_DEVICE_REMOVE_PEND = 1          ' Éè±¸,½«Òª°Î³ö
Public Const USBIO_DEVICE_REMOVE = 0               ' Éè±¸,°Î³öÊÂ¼þ,ÒÑ¾°Î³ö

Declare Function USBIO_SetDeviceNotify Lib "USBIOX.DLL" (ByVal iIndex As Long, ByRef iDeviceID As String, ByVal
iNotifyRoutine As Long) As Boolean
' USBIO_SetDeviceNotify   Éè¶¨Éè±¸ÊÂ¼þÍ¨Öª³ÌĐò
' iIndex,              Ö¸¶¨USB2ISPÉè±¸Đò°Å,0¶ÔÓ¦µÚÒ»¸öÉè±¸,
' iDeviceID,            ¿ÉÑ¡²ÎÊý,Ö¸Ïò×Ô·û´®,Ö¸¶¨±»¼à¿ØµÄÉè±¸,µÄID,×Ö·û´®ÒÔ\0ÖÕÖ¹
' iNotifyRoutine        (°ü¯ÊýµØÖ·)Ö¸¶¨Éè±¸ÊÂ¼þ»Øµ÷³ÌĐò, Î³NULLÔòÈ¡ÏûÊÂ¼þÍ¨Öª, ·ñÔòÔÙ¼ì²âµ½ÊÂ¼þÊ±µ÷ÓÃ,Ã³ÌĐò
```

ANEXO B - frmMain modificado para leitura única

Form frmMain

```
Option Explicit

Dim hopen As Long

Private Sub Command1_Click()

  Dim iBuff As arrRBuffer
  Dim buffer As arrRBuffer

  'mWRLen = HexToBcd(I2CWRLen.Text)
  'mRdLen = HexToBcd(I2CRDLen.Text)

  'Registrador 0
  Call mStrtoVal("58000000B2", buffer, "5")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

  If (mOpen = True) Then
   If (USBIO_StreamI2C(mIndex, "5", buffer, "0", iBuff) = False) Then
     MsgBox "Erro na escrita do registrador 0", vbExclamation, "USB2I2C DEMO"
   End If
  End If

  'Leitura do registrador 8
  Call mStrtoVal("580008", buffer, "3")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

  If (mOpen = True) Then
   If (USBIO_StreamI2C(mIndex, "3", buffer, "0", iBuff) = False) Then
     MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
   End If
  End If



  Call mStrtoVal("59", buffer, "1")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

  If (mOpen = True) Then
   If (USBIO_StreamI2C(mIndex, "1", buffer, "2", iBuff) = False) Then
     MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
   Else

      Dim buff As String
      Dim i As Long
      For i = 0 To 2 - 1
       buff = buff & Hex2bit(iBuff.buf(i)) + " "
      Next
      I2CRDBuf.Text = buff

   End If
  End If
```

```
'Leitura do registrador 9
Call mStrtoVal("580009", buffer, "3")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
 If (USBIO_StreamI2C(mIndex, "3", buffer, "0", iBuff) = False) Then
   MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
 End If
End If



Call mStrtoVal("59", buffer, "1")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
 If (USBIO_StreamI2C(mIndex, "1", buffer, "2", iBuff) = False) Then
   MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
 Else

    For i = 0 To 2 - 1
     buff = buff & Hex2bit(iBuff.buf(i)) + " "
    Next
    I2CRDBuf.Text = buff

 End If
End If

'Leitura do registrador A
Call mStrtoVal("58000A", buffer, "3")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
 If (USBIO_StreamI2C(mIndex, "3", buffer, "0", iBuff) = False) Then
   MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
 End If
End If



Call mStrtoVal("59", buffer, "1")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
 If (USBIO_StreamI2C(mIndex, "1", buffer, "2", iBuff) = False) Then
   MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
 Else

    For i = 0 To 2 - 1
     buff = buff & Hex2bit(iBuff.buf(i)) + " "
    Next
    I2CRDBuf.Text = buff

 End If
End If



'Registrador 2
Call mStrtoVal("5800023230", buffer, "5")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý
```

```
If (mOpen = True) Then
  If (USBIO_StreamI2C(mIndex, "5", buffer, "0", iBuff) = False) Then
    MsgBox "Erro na escrita do registrador 0", vbExclamation, "USB2I2C DEMO"
  End If
End If

'Registrador 3
Call mStrtoVal("5800030819", buffer, "5")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
  If (USBIO_StreamI2C(mIndex, "5", buffer, "0", iBuff) = False) Then
    MsgBox "Erro na escrita do registrador 0", vbExclamation, "USB2I2C DEMO"
  End If
End If

'Registrador 4
Call mStrtoVal("5800040832", buffer, "5")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
  If (USBIO_StreamI2C(mIndex, "5", buffer, "0", iBuff) = False) Then
    MsgBox "Erro na escrita do registrador 0", vbExclamation, "USB2I2C DEMO"
  End If
End If

'Registrador 5
Call mStrtoVal("5800050000", buffer, "5")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
  If (USBIO_StreamI2C(mIndex, "5", buffer, "0", iBuff) = False) Then
    MsgBox "Erro na escrita do registrador 0", vbExclamation, "USB2I2C DEMO"
  End If
End If

'Registrador 6
Call mStrtoVal("5800060000", buffer, "5")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
  If (USBIO_StreamI2C(mIndex, "5", buffer, "0", iBuff) = False) Then
    MsgBox "Erro na escrita do registrador 0", vbExclamation, "USB2I2C DEMO"
  End If
End If

'Registrador 7
Call mStrtoVal("5800070001", buffer, "5")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
  If (USBIO_StreamI2C(mIndex, "5", buffer, "0", iBuff) = False) Then
    MsgBox "Erro na escrita do registrador 0", vbExclamation, "USB2I2C DEMO"
  End If
End If

'Registrador 1
Call mStrtoVal("5800010FFF", buffer, "5")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

If (mOpen = True) Then
  If (USBIO_StreamI2C(mIndex, "5", buffer, "0", iBuff) = False) Then
```

```
      MsgBox "Erro na escrita do registrador 0", vbExclamation, "USB2I2C DEMO"
    End If
  End If

  'Leitura do registrador 8
  Call mStrtoVal("580008", buffer, "3")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

  If (mOpen = True) Then
    If (USBIO_StreamI2C(mIndex, "3", buffer, "0", iBuff) = False) Then
      MsgBox "I2CÁ÷Ä£ÊÍ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
    End If
  End If



  Call mStrtoVal("59", buffer, "1")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

  If (mOpen = True) Then
    If (USBIO_StreamI2C(mIndex, "1", buffer, "2", iBuff) = False) Then
      MsgBox "I2CÁ÷Ä£ÊÍ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
    Else

      For i = 0 To 2 - 1
        buff = buff & Hex2bit(iBuff.buf(i)) + " "
      Next
      I2CRDBuf.Text = buff

    End If
  End If


  'Leitura do registrador 9
  Call mStrtoVal("580009", buffer, "3")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

  If (mOpen = True) Then
    If (USBIO_StreamI2C(mIndex, "3", buffer, "0", iBuff) = False) Then
      MsgBox "I2CÁ÷Ä£ÊÍ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
    End If
  End If



  Call mStrtoVal("59", buffer, "1")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

  If (mOpen = True) Then
    If (USBIO_StreamI2C(mIndex, "1", buffer, "2", iBuff) = False) Then
      MsgBox "I2CÁ÷Ä£ÊÍ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
    Else

      For i = 0 To 2 - 1
        buff = buff & Hex2bit(iBuff.buf(i)) + " "
      Next
      I2CRDBuf.Text = buff

  End If
```

```
 End If

 'Leitura do registrador A
 Call mStrtoVal("58000A", buffer, "3")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª»»ÊÊÝÖµÊý¾Ý

 If (mOpen = True) Then
  If (USBIO_StreamI2C(mIndex, "3", buffer, "0", iBuff) = False) Then
    MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
  End If
 End If


 Call mStrtoVal("59", buffer, "1")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª»»ÊÊÝÖµÊý¾Ý

 If (mOpen = True) Then
  If (USBIO_StreamI2C(mIndex, "1", buffer, "2", iBuff) = False) Then
    MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
  Else

     For i = 0 To 2 - 1
       buff = buff & Hex2bit(iBuff.buf(i)) + " "
     Next
     I2CRDBuf.Text = buff

  End If
 End If

 'Configurando os registrados de CIN3

 'Registrador 98
 Call mStrtoVal("5800983FBF", buffer, "5")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª»»ÊÊÝÖµÊý¾Ý

 If (mOpen = True) Then
  If (USBIO_StreamI2C(mIndex, "5", buffer, "0", iBuff) = False) Then
    MsgBox "Erro na escrita do registrador 0", vbExclamation, "USB2I2C DEMO"
  End If
 End If

 'Registrador 99
 Call mStrtoVal("5800999FFF", buffer, "5")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª»»ÊÊÝÖµÊý¾Ý

 If (mOpen = True) Then
  If (USBIO_StreamI2C(mIndex, "5", buffer, "0", iBuff) = False) Then
    MsgBox "Erro na escrita do registrador 0", vbExclamation, "USB2I2C DEMO"
  End If
 End If

 'Registrador 9A
 Call mStrtoVal("58009A00A0", buffer, "5")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª»»ÊÊÝÖµÊý¾Ý

 If (mOpen = True) Then
  If (USBIO_StreamI2C(mIndex, "5", buffer, "0", iBuff) = False) Then
    MsgBox "Erro na escrita do registrador 0", vbExclamation, "USB2I2C DEMO"
  End If
 End If
```

```
    'Configurando os registrados de CIN9

    'Registrador C8
    Call mStrtoVal("5800C83FFF", buffer, "5")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

    If (mOpen = True) Then
      If (USBIO_StreamI2C(mIndex, "5", buffer, "0", iBuff) = False) Then
        MsgBox "Erro na escrita do registrador 0", vbExclamation, "USB2I2C DEMO"
      End If
    End If

    'Registrador C9
    Call mStrtoVal("5800C99FEF", buffer, "5")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

    If (mOpen = True) Then
      If (USBIO_StreamI2C(mIndex, "5", buffer, "0", iBuff) = False) Then
        MsgBox "Erro na escrita do registrador 0", vbExclamation, "USB2I2C DEMO"
      End If
    End If

    'Registrador CA
    Call mStrtoVal("5800CA00A0", buffer, "5")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

    If (mOpen = True) Then
      If (USBIO_StreamI2C(mIndex, "5", buffer, "0", iBuff) = False) Then
        MsgBox "Erro na escrita do registrador 0", vbExclamation, "USB2I2C DEMO"
      End If
    End If

End Sub


Private Sub Command2_Click()

Dim iBuff As arrRBuffer
Dim buffer As arrRBuffer
'Leitura do registrador E
    Call mStrtoVal("58000E", buffer, "3")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

    If (mOpen = True) Then
      If (USBIO_StreamI2C(mIndex, "3", buffer, "0", iBuff) = False) Then
        MsgBox "I2CÁ÷Ä£ÊÍ½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
      End If
    End If




    Call mStrtoVal("59", buffer, "1")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

    If (mOpen = True) Then
      If (USBIO_StreamI2C(mIndex, "1", buffer, "2", iBuff) = False) Then
        MsgBox "I2CÁ÷Ä£ÊÍ½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
      Else

        Dim buff As String
        Dim i As Long
```

```
        For i = 0 To 2 - 1
          buff = buff & Hex2bit(iBuff.buf(i)) + ""
        Next
        I2CRDBuf.Text = buff
        Dim intFile As Integer
        Dim strFile As String
        strFile = "C:\Users\Public\CIN3.txt"
        intFile = FreeFile
        Open strFile For Input As #intFile
        Dim FiletoStr As String
        FiletoStr = StrConv(InputB(LOF(intFile), intFile), vbUnicode)
        Close #intFile
        Open strFile For Output As #intFile
        Print #intFile, FiletoStr + CStr(HexToDec(buff)) + " ";
        Close #intFile
      End If
    End If

    'Leitura do registrador 14
    Dim iBuff2 As arrRBuffer
    Dim buffer2 As arrRBuffer

    Call mStrtoVal("580014", buffer2, "3")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

    If (mOpen = True) Then
      If (USBIO_StreamI2C(mIndex, "3", buffer2, "0", iBuff2) = False) Then
        MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
      End If
    End If



    Call mStrtoVal("59", buffer2, "1")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

    If (mOpen = True) Then
      If (USBIO_StreamI2C(mIndex, "1", buffer2, "2", iBuff2) = False) Then
        MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
      Else
        Dim buff2 As String
        For i = 0 To 2 - 1
          buff2 = buff2 & Hex2bit(iBuff2.buf(i)) + ""
        Next
        I2CRDBuf.Text = buff2
        strFile = "C:\Users\Public\CIN9.txt"
        intFile = FreeFile
        Open strFile For Input As #intFile
        FiletoStr = StrConv(InputB(LOF(intFile), intFile), vbUnicode)
        Close #intFile
        Open strFile For Output As #intFile
        Print #intFile, FiletoStr + CStr(HexToDec(buff2)) + " ";
        Close #intFile
      End If
    End If
End Sub


Private Sub Command3_Click()
  Dim iBuff As arrRBuffer
```

```
Dim buffer As arrRBuffer
'Leitura do registrador E
  Call mStrtoVal("58000E", buffer, "3")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

  If (mOpen = True) Then
   If (USBIO_StreamI2C(mIndex, "3", buffer, "0", iBuff) = False) Then
     MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
   End If
  End If



  Call mStrtoVal("59", buffer, "1")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

  If (mOpen = True) Then
   If (USBIO_StreamI2C(mIndex, "1", buffer, "2", iBuff) = False) Then
     MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
   Else

      Dim buff As String
      Dim i As Long
      For i = 0 To 2 - 1
       buff = buff & Hex2bit(iBuff.buf(i)) + ""
      Next
      I2CRDBuf.Text = buff
      Dim intFile As Integer
      Dim strFile As String
      strFile = "C:\Users\Luana Ruiz\workspace\CIN3single.txt"
      intFile = FreeFile
      Open strFile For Output As #intFile
      Print #intFile, CStr(HexToDec(buff)) + " ";
      Close #intFile
   End If
  End If

  'Leitura do registrador 14
  Dim iBuff2 As arrRBuffer
  Dim buffer2 As arrRBuffer

  Call mStrtoVal("580014", buffer2, "3")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

  If (mOpen = True) Then
   If (USBIO_StreamI2C(mIndex, "3", buffer2, "0", iBuff2) = False) Then
     MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
   End If
  End If



  Call mStrtoVal("59", buffer2, "1")      '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

  If (mOpen = True) Then
   If (USBIO_StreamI2C(mIndex, "1", buffer2, "2", iBuff2) = False) Then
     MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
   Else
      Dim buff2 As String
      For i = 0 To 2 - 1
```

```
            buff2 = buff2 & Hex2bit(iBuff2.buf(i)) + ""
          Next
          I2CRDBuf.Text = buff2
          strFile = "C:\Users\Luana Ruiz\workspace\CIN9single.txt"
          intFile = FreeFile
          Open strFile For Output As #intFile
          Print #intFile, CStr(HexToDec(buff2)) + " ";
          Close #intFile
        End If
      End If
    End Sub


Private Sub eepromRdDate_Click()
 Dim mDataAddr As Long
 Dim mLen As Long
 Dim buffer As arrRBuffer
 Dim bu() As Byte
 mLen = HexToBcd(RdDataLen)

 If (RdDataAddr.Text = "") Then
  MsgBox "ÇëÊäÈëÊý¾Ýµ¥ÔªÆðÊ¼µØÖ·£¡", vbExclamation, "USB2I2C DEMO"
  Exit Sub
 End If
 If (mLen <= 0) Then
  MsgBox "ÇëÊäÈë¶ÁÈ¡³¤¶È£¡", vbExclamation, "USB2I2C DEMO"
  Exit Sub
 End If
 mDataAddr = HexToBcd(RdDataAddr)
 If (mOpen = True) Then
   If (USBIO_ReadEEPROM(mIndex, eepromid, mDataAddr, mLen, buffer)) Then
     Dim buff As String
     Dim i As Long
     For i = 0 To mLen - 1
       buff = buff & Hex2bit(buffer.buf(i)) & " "
     Next i
     RdDataBuf.Text = buff
   Else
     MsgBox "¶ÁE2PROMÊý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
   End If
   RdDataLen.Text = Hex(mLen)
 Else
   MsgBox "USB2I2C Device NOT Oper!", vbExclamation, "USB2I2C DEMO"
 End If
End Sub

Private Sub eepromWrDate_Click()
 Dim mData As Byte
 Dim mDataAddr As Long
 Dim mLen As Long
 Dim buffer As arrRBuffer

 mLen = HexToBcd(WrDataLen.Text)
 If (WrDataAddr.Text = "") Then
  MsgBox "ÇëÊäÈëÊý¾Ýµ¥ÔªÆðÊ¼µØÖ·£¡", vbExclamation, "USB2I2C DEMO"
  Exit Sub
 End If
 If (mLen <= 0 Or WrDataBuf.Text = "") Then
  MsgBox "ÇëÊäÈëÒª´Èëµ¥ÔªÊý¾Ý,³¤¶È£¡", vbExclamation, "USB2I2C DEMO"
  Exit Sub
```

```
  End If

  If (mLen > (Len(WrDataBuf) \ 2)) Then 'ÔÚÊäÈë³¤¶È°ÍÊý¾Ý³¤¶ÈÖÐÈ¡Ð¡Öµ
    mLen = Len(WrDataBuf) \ 2
  End If

  mDataAddr = HexToBcd(WrDataAddr.Text)
  Call mStrtoVal(WrDataBuf.Text, buffer, mLen)  '½«ÊäÈëµÄÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊýÖµÊý¾Ý

  If (mOpen = True) Then
    If (USBIO_WriteEEPROM(mIndex, eepromid, mDataAddr, mLen, buffer) = False) Then
      MsgBox "¶ÁE2PROMÊý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
    End If
    WrDataLen.Text = Hex(mLen)
  Else
    MsgBox "USB2I2C Device NOT Oper!", vbExclamation, "USB2I2C DEMO"
  End If
End Sub

Private Sub eppromtype_Click(Index As Integer)
Select Case Index
    Case 0
      eepromid = ID_24C01
    Case 1
      eepromid = ID_24C02
    Case 2
      eepromid = ID_24C04
    Case 3
      eepromid = ID_24C08
    Case 4
      eepromid = ID_24C16
    Case 5
      eepromid = ID_24C32
    Case 6
      eepromid = ID_24C64
    Case 7
      eepromid = ID_24C128
    Case 8
      eepromid = ID_24C256
    Case 9
      eepromid = ID_24C512
    Case 10
      eepromid = ID_24C1024
    Case 11
      eepromid = ID_24C2048
    Case 12
      eepromid = ID_24C4096
 End Select
End Sub

Private Sub Form_Load()
mIndex = 0
   SSTab1.TabVisible(0) = False
   SSTab1.TabVisible(1) = False
   SSTab1.TabVisible(2) = False
   SSTab1.TabVisible(3) = False
   SSTab1.TabVisible(4) = False
   SSTab1.TabVisible(5) = False
hopen = USBIO_OpenDevice(mIndex)
If (hopen = INVALID_HANDLE_VALUE) Then
```

```
    mOpen = False
  Else
    mOpen = True
  End If
  'ÉèÖÃÉè±¸²â°ÎÏ¨Öª
  If USBIO_SetDeviceNotify(mIndex, vbNullString, AddressOf mUSBIO_NOTIFY_ROUTINE) = False Then
    MsgBox "ÉèÖÃÉè±¸²â°ÎÏ¨Öª£§°Ü", vbExclamation, "USB2I2C DEMO"
  End If
  enablebtn (mOpen)

  'Executar aqui a configuração inicial
  'Executar também a leitura única

  Call Command1_Click
  Call Command3_Click

  'Unload Me

  'End

  End Sub

  Private Sub Form_Unload(Cancel As Integer)
  USBIO_SetDeviceNotify mIndex, vbNullString, 0&
  If (mOpen = True) Then
    USBIO_CloseDevice (mIndex)

  End If
  End Sub



  Private Sub StreamICRW_Click()
  Dim mWRLen As Long
  Dim mRdLen As Long
  Dim iBuff As arrRBuffer
  Dim buffer As arrRBuffer

  mWRLen = HexToBcd(I2CWRLen.Text)
  mRdLen = HexToBcd(I2CRDLen.Text)

  '----------------------------------------
  If (I2CM(0).Value = True) Then
    If (USBIO_SetStream(mIndex, &H80) = False) Then
      MsgBox "ÉèÖÃI2CÊ±ÖÓ = 20KHzÊ§°Ü£¡ ", vbExclamation, "USB2I2C DEMO"
      Exit Sub
    End If
  ElseIf (I2CM(1).Value = True) Then
    If (USBIO_SetStream(mIndex, &H81) = False) Then
      MsgBox "ÉèÖÃI2CÊ±ÖÓ = 100KHzÊ§°Ü£¡ ", vbExclamation, "USB2I2C DEMO"
      Exit Sub
    End If
  ElseIf (I2CM(2).Value = True) Then
    If (USBIO_SetStream(mIndex, &H82) = False) Then
      MsgBox "ÉèÖÃI2CÊ±ÖÓ = 400KHzÊ§°Ü£¡ ", vbExclamation, "USB2I2C DEMO"
      Exit Sub
    End If
  ElseIf (I2CM(3).Value = True) Then
    If (USBIO_SetStream(mIndex, &H83) = False) Then
```

```
        MsgBox "ÉèÖÃI2CÊ±ÖÓ = 750KHzÊ§°Ü£¡ ", vbExclamation, "USB2I2C DEMO"
        Exit Sub
      End If
    End If
    '----------------------------------------


    If (mWRLen > 0 And I2CWRBuf.Text = "") Then
     MsgBox "ÇëÊäÈëÒª´µÄÊý¾Ý,³¤¶ÈÊ£¡", vbExclamation, "USB2I2C DEMO"
     Exit Sub
    End If
    If ((mWRLen = 0) And (mRdLen = 0)) Then
     MsgBox "ÇëÊäÈë¶ÁÊý¾ÝËùÐèµÄ³¤¶ÈÊ£¡", vbExclamation, "USB2I2C DEMO"
     Exit Sub
    End If
    If (mWRLen > Len(Trim(I2CWRBuf.Text)) \ 2) Then
      mWRLen = Len(Trim(I2CWRBuf.Text)) \ 2
    End If

    Call mStrtoVal(I2CWRBuf.Text, buffer, mWRLen)      '½«ÊäÈëµÄÂÊ®Áù½øÖÆ¸ñÊ½×Ö·ûÊý¾Ý×ª³ÉÊý¾ÝÊý¾Ý

    If (mOpen = True) Then
     If (USBIO_StreamI2C(mIndex, mWRLen, buffer, mRdLen, iBuff) = False) Then
       MsgBox "I2CÁ÷Ä£Ê½¶ÁÐ´Êý¾ÝÊ§°Ü£¡", vbExclamation, "USB2I2C DEMO"
     Else
      If (mRdLen > 0) Then   'ÓÐÊý¾Ý·µ»Ø
        Dim buff As String
        Dim i As Long
        For i = 0 To mRdLen - 1
          buff = buff & Hex2bit(iBuff.buf(i)) + " "
        Next
        I2CRDBuf.Text = buff
      End If
     End If
     I2CWRLen.Text = Hex(mWRLen)
     I2CRDLen.Text = Hex(mRdLen)
    Else
     MsgBox "USB2I2C Device NOT Oper!", vbExclamation, "USB2I2C DEMO"
    End If
    End Sub

    Private Sub USBIO_NOTIFY_ROUTINE_KeyUp(KeyCode As Integer, Shift As Integer)  'Éè±¸²å°ÎÍ¨Ö¸¦ÀÍ³ÌÐò
      Dim iEventStatus As Long
      iEventStatus = KeyCode '²å°ÎÊÂ¼þ
      If (iEventStatus = USBIO_DEVICE_ARRIVAL) Then ' Éè±¸²åÈëÊÂ¼þ,ÒÑ¾²åÈë
        If (USBIO_OpenDevice(mIndex) = INVALID_HANDLE_VALUE) Then
          MsgBox "´ò¿ªÉè±¸Ê§°Ü!", vbOK, "USB2I2C DEMO"
          mOpen = False
        Else
          mOpen = True  '´ò¿ª³É¹¦
        End If
      ElseIf (iEventStatus = USBIO_DEVICE_REMOVE) Then ' Éè±¸³ö²ÊÂ¼þ,ÒÑ¾³ö²ö
        USBIO_CloseDevice (mIndex)
        mOpen = False
      End If
      enablebtn (mOpen) 'Éè±¸´ò¿ª,°´Â¥¿ÉÓÃ,Éè±¸Ã»´ò¿ª,°´Â¥½ûÓÃ
    End Sub

    Public Sub enablebtn(ByVal bEnable As Boolean)  'bEnable=true :¸÷°ìå°´Â¥¿ÉÓÃ ;=false:enable:¸÷°ìå°´Â¥½ûÓÃ
     With frmMain
```

```
    .StreamICRW.Enabled = bEnable

    .eepromRdDate.Enabled = bEnable
    .eepromWrDate.Enabled = bEnable

    If (bEnable = True) Then '°ìå±êÌâÏÔÊ¾
       frmMain.Caption = "USB2I2C **PlugIn"

    Else
       frmMain.Caption = "USB2I2C **PlugOut"

    End If
  End With


End Sub
```

ANEXO C - Código do GUI desenvolvido

```
function varargout = interface2(varargin)
% INTERFACE MATLAB code for interface.fig
%      INTERFACE, by itself, creates a new INTERFACE or raises the existing
%      singleton*.
%
%      H = INTERFACE returns the handle to a new INTERFACE or the handle to
%      the existing singleton*.
%
%      INTERFACE('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in INTERFACE.M with the given input arguments.
%
%      INTERFACE('Property','Value',...) creates a new INTERFACE or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before interface_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to interface_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help interface

% Last Modified by GUIDE v2.5 04-Nov-2016 15:37:19

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @interface_OpeningFcn, ...
                   'gui_OutputFcn',  @interface_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT



% --- Executes just before interface is made visible.
function interface_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to interface (see VARARGIN)

polegar = load('polegarfinal.mat', '-mat');
```

```
polegar_cell = struct2cell(polegar);
handles.polegar_mat = cell2mat(polegar_cell);
indicador = load('indicadorfinal.mat', '-mat');
indicador_cell = struct2cell(indicador);
handles.indicador_mat = cell2mat(indicador_cell);


handles.trainedClassifier = varargin{1};

result = load('resultadofinal.mat', '-mat');
result_cell = struct2cell(result);
handles.result_mat = cell2mat(result_cell);
for i = 1:length(handles.result_mat)
    if handles.result_mat(i, 1) == 0 %caderno
        handles.colors(i, 1) = 255;
        handles.colors(i, 2) = 0;
        handles.colors(i, 3) = 0;
    end
    if handles.result_mat(i, 1) == 1 %pelucia
        handles.colors(i, 1) = 0;
        handles.colors(i, 2) = 0;
        handles.colors(i, 3) = 255;
    end
    if handles.result_mat(i, 1) == 2 %garrafa cheia
        handles.colors(i, 1) = 0;
        handles.colors(i, 2) = 255;
        handles.colors(i, 3) = 0;
    end
    if handles.result_mat(i, 1) == 3 %garrafa meio cheia, amarelo
        handles.colors(i, 1) = 255;
        handles.colors(i, 2) = 255;
        handles.colors(i, 3) = 0;
    end
     if handles.result_mat(i, 1) == 4 %garrafa vazia, turquesa
        handles.colors(i, 1) = 0;
        handles.colors(i, 2) = 128;
        handles.colors(i, 3) = 127;
    end
end
handles.current_x = handles.polegar_mat;
handles.current_y = handles.indicador_mat;
scatter(handles.current_x, handles.current_y, 3, handles.colors, 'filled');
% Choose default command line output for interface
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes interface wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = interface_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Get default command line output from handles structure
varargout{1} = handles.output;



% --- Executes on button press in classificar.
function classificar_Callback(hObject, eventdata, handles)
% hObject    handle to classificar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

handles.answer = handles.trainedClassifier.predict([handles.pol, handles.ind]);

if handles.answer == 0
   set(handles.display,'String', 'Caderno');
end
if handles.answer == 1
   set(handles.display,'String', 'Pelúcia');
end
if handles.answer == 2
   set(handles.display,'String', 'Garrafa cheia');
end
if handles.answer == 3
   set(handles.display,'String', 'Garrafa meio cheia');
end
if handles.answer == 4
   set(handles.display,'String', 'Garrafa vazia');
end
guidata(hObject, handles);

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in I2CBus.
function I2CBus_Callback(hObject, eventdata, handles)
% hObject    handle to I2CBus (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
system('USB2I2C_DEMO_VBEN')

fileID = fopen('CIN9single.txt');
handles.pol = fscanf(fileID, '%i');
fclose(fileID);

fileID = fopen('CIN3single.txt');
handles.ind = fscanf(fileID, '%i');
fclose(fileID);

set(handles.display_pol,'String', handles.pol);

set(handles.display_ind,'String', handles.ind);

guidata(hObject, handles);
```

ANEXO D - Artigo escrito para a *International Symposium on Consumer Electronics* (ISCE) 2016

Comparative Analysis of Classification Algorithms on Tactile Sensors

Wesley Becari, Luana Ruiz, Bruno G. P. Evaristo, Francisco J. R. Fernandez
Department of Electronic System Engineering
Polytechnic School of the University of São Paulo
São Paulo, Brazil
wesley@lme.usp.br, luana.ruiz@usp.br, brunopacci@gmail.com, jramirez@lme.usp.br

***Abstract*—A comparative analysis of classification algorithms of iCub platform humanoid hand tactile sensors is presented. The experimental data were analyzed with different learning supervised classification algorithms: Decision Trees Classifiers, *k*-Nearest Neighbors Classifiers (kNN), Ensemble Classifiers, and Support Vector Machines (SVM). The best result was obtained with a Gaussian SVM kernel, which allowed 97.4% accuracy using 20% data for holdout validation. The results indicate the potential of categorization and learning of robotic hands for object grasping and manipulation.**

Keywords— Tactile sensors; Classification algorithms; Haptic interfaces; Machine learning algorithms.

I. Introduction

Tactile sensors are employed to sense stimuli from direct physical contact for examining, identifying, and manipulating objects in the environment [1]. Tactile information allows examining the weight, slipperiness, texture and hardness of objects. Different transduction methods can be used for tactile sensors: capacitive, optical, piezoresistive, magnetic, and piezoelectric.

Robotic applications need effective tactile sensors; however, these sensors are still undeveloped as compared to computer vision. The iCub platform humanoid offers a platform to investigate the tactile sensors interaction with objects [2]. With its physical interaction, it is possible to recognize texture and objects using classification algorithms.

The hand of the iCub has five fingers with 12 sensitive zones. The fingertip incorporates a capacitive pressure sensor. When pressure is applied to the surface of the fingertip, the capacitance changes accordingly. The change in capacitance is an estimation of the pressure applied to the sensor. The round pads are connected to a capacitance to digital converter (CDC) AD7147. The CDC provides 12 measurements of capacitance for each finger, with values ranging from 0 to 255 [3].

A dataset of nine different everyday objects is available online [3]. This paper presents a comparative analysis of iCub grasping motion dataset using different classification algorithms to recognize the objects. Section II presents the classification algorithms. Section III describes the dataset, and Section IV demonstrates the offline classification results. The conclusions are described in Section V.

II. Classification Algorithms

A. Decision trees

Decision trees can be used for sequential decision problems. This method divides the data domain into multiple spaces, mapping them to predict responses. The hierarchical structure of decision trees consists of three types of nodes: root node, internal nodes, and terminal (leaf) nodes (class label). Starting from the root node, "if-then" tests are applied to each sample, following the appropriate branch based on the splitting criteria of the test [4]. The tests will be repeated for internal nodes, increasing the number of sub-spaces. The terminal node will be associated with a class label. The algorithm stops when stopping are is satisfied or when the number of splits defined is reached.

B. Support Vector Machines

The SVM is a kernel-based methodology, which depends only on dot-products. This method can generate non-linear decision boundaries [4]. The SVM constructs a decision function to maximize the margin between two different classes. The simplest example occurs when the dataset is linearly separable. In this case, a hyperplane can divide the data, allowing mapping an input pattern to a response class. For data not linearly separable, it is possible generate non-linear decision boundaries with non-linear kernel functions, such as quadratic, cubic, and Gaussian.

C. k-Nearest Neighbor classifiers

The kNN is a classification algorithm based on the proximity of the values of some attributes. Samples with

similar characteristics appear as clustered points. A common distance function is based on the Euclidean distance. New samples compute the distances between all previous data, already classified into clusters. The kNN algorithm will select the *k*-nearest neighbors presenting the smallest distance values and the new instance will be added to the largest cluster [4].

D. Ensemble classifiers

Ensemble methods combine the predictions of multiple classifiers, such as Decision Trees or Neural Networks, instead of learning from a single classifier. With the ensemble, one classifier compensates the errors made by another, reducing the variance and the bias [4]. Bagging and Boosting are two methods for producing ensembles. Bagging separates samples of the training dataset and trains a classifier for each subset. The results are combined with the average or the majority voting of the predicted class. Differently, Boosting creates subsets by re-sampling the training patterns.

III. iCub Grasping Dataset Analysis

The iCub grasping measurements were performed with fixed position and orientation. The hands digits were fully opened. The objects for grasping tests were placed in this pre-shape position. The controller closed the digits until the tactile sensors reached a critical threshold [3].

The tactile sensor dataset was obtained from nine objects: empty plastic bottle; half-full plastic bottle; full plastic bottle; empty can; half-full can; teddy bear; monkey soft-toy; hardcover book; bottle of lotion. Each object was grasped 20 times with the 12 tactile sensors. For the comparative analysis, only three fingers were enough to classify the objects (thumb, index, and medium finger).

The classification algorithms were trained with Matlab. The validation set, used to determine the accuracy of the models, was obtained by splitting 20% of the dataset.

The Decision Trees were based on the Gini criteria. Three degrees of splitting were tested: complex (tree with 100 splits), medium (tree with 20 splits), and simple (tree with 4 splits). Four SVM kernels were analyzed: linear, quadratic, cubic, and Gaussian. The kNN algorithm was configured with Euclidean distance metric and 1 neighbor (Fine kNN), 10 neighbors (Medium kNN), 100 neighbors (Coarse kNN). The Cubic kNN was formed with 10 neighbors and Minkowski distance metric. The Boosted Trees and Bagged Trees were configured with AdaBoost learner type, 30 learners, and learning rate of 0.2.

IV. Results and Discussion

Table 1 presents the comparative performance of the classification algorithms.

TABLE I. Comparative Performance of Classification Algorithms

Classification Algorithms	3 fingers (Thumb, Index, and Medium) (%)	2 Fingers (Thumb and Index) (%)	2 Fingers (Thumb and Medium) (%)	1 Finger (Thumb) (%)
Complex Tree	94.3	81.3	87.5	71.5
Medium Tree	92.4	81.3	86.6	75.0
Simple Tree	59.2	55.8	59.4	56.6
Linear SVM	90.8	78.5	85.5	68.4
Quadratic SVM	94.2	83.0	88.6	62.9
Cubic SVM	94.3	81.3	88.0	54.8
Gaussian SVM	97.4	85.1	89.0	74.3
Fine kNN	96.1	82.0	85.8	49.7
Medium kNN	94.6	85.0	88.5	68.4
Coarse kNN	85.3	75.6	83.1	72.1
Cubic kNN	94.4	85.1	88.5	68.4
Boosted Trees	94.0	81.3	87.1	74.6
Bagged Trees	95.7	82.8	86.6	70.5

The highest percentage of correct answers was achieved using SVM. Good results were verified with Fine and Medium kNN. Ensemble Trees (Boosted and Bagged) presented more accurate classification than single tree classifiers, as expected.

For example, the scatter graph with the nine objects tested with Gaussian SVM is presented in Fig.1. An accuracy of 85.1% was obtained with the thumb and index finger. Misclassified data occurs mainly in the boundaries of clusters. Fig. 2 presents the evaluation of the performance of the classification model, using a confusion matrix.

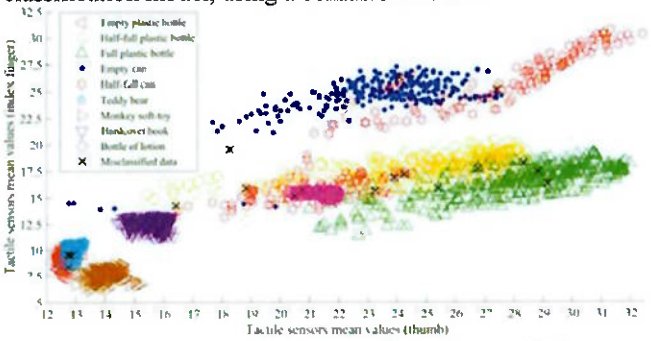


Fig. 1. Scatter graph of movements tests using the Gaussian SVM kernel.

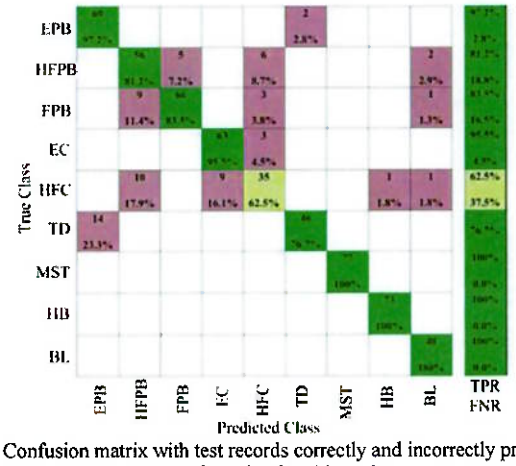


Fig. 2. Confusion matrix with test records correctly and incorrectly predicted by the Gaussian SVM kernel.

V. Conclusion

Different classifiers were trained and tested with iCub tactile sensors grasping data. The comparative analysis shows that the Gaussian SVM classifier can be effectively used with

97.4% accuracy using three fingers. The results suggest a great potential to create an intelligent system capable of sensing materials, weight, textures, and hardness.

References

[1] J. Lazar, H. J. Feng, and H. Hochheiser. "Research methods in Human Computer Interaction." West Sussex: John Wiley & Sons, 2010.

[2] G. Metta, G. Sandini, D. Vernon, L. Natale, and F. Nori, "The iCub humanoid robot: an open platform for research in embodied cognition," in 8th Work. on Performance Metrics for Intelligent Systems, 2008.

[3] H. Soh, Y. Su, and Y. Demiris, "Online Spatio-Temporal Gaussian Process Experts with Application to Tactile Classification", IEEE/RSJ International Conference on Intelligent Robots and Systems, 2012.

[4] C. C. Aggarwal et al. "Data Classification: Algorithms and Applications," CRC Press, 2014.