ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

SABRINA NABUCO DE OLIVEIRA

CONTROLE DE MOVIMENTAÇÃO DE UM VEÍCULO SUBAQUÁTICO

SÃO PAULO

2016

SABRINA NABUCO DE OLIVEIRA

CONTROLE DE MOVIMENTAÇÃO DE UM VEÍCULO SUBAQUÁTICO

Relatório final, apresentado a Escola Politécnica da USP, como parte das exigências para a obtenção do título de Engenheira Eletricista.

Orientador: Prof. Dr. Felipe Pait
Coordenador: Prof. Dr. Fuad Kassab

SÃO PAULO
2016

Dedico este trabalho aos meus pais, familiares e amigos, que sempre acreditam nos meus sonhos e me apoiam para realizar minhas conquistas na vida.

AGRADECIMENTOS

Aos meus pais por sempre acreditarem em mim e fornecerem incondicionalmente todo apoio estrutural para que eu pudesse prosseguir nos estudos.

Aos meus irmãos, familiares e amigos.

Ao meu orientador, Prof. Dr. Felipe Pait, pelo acompanhamento, orientação e amizade.

Ao meu coordenador, Prof. Dr. Fuad Kassab, pelo apoio, disponibilidade e confiança.

Ao Anselmo por todo suporte e assistência fornecidos.

À OBMEP por ter estado presente na minha vida como incentivo.

Às jovens e aos jovens negros e periféricos que infelizmente não tiveram a mesma oportunidade que eu tive para realizar esse curso.

Aos meus colegas e professores da rede pública que sempre me apoiaram e me incentivaram nos estudos.

“Eu sou definida como a outra em todos os grupos de que participo. A forasteira, tanto pela força como pela fraqueza"

(Audre Lorde)

RESUMO

Este trabalho consiste na modelagem e análises hidrodinâmicas de um ROV, projeto de controladores que possam levar o ROV a estados desejados, uso de eletrônica capaz de possibilitar a movimentação e controle do ROV, a troca de toda essa eletrônica inicial, motores, hélices, sensores, proteções entre eletrônica de controle e motores, desenvolvimento de um joystick e a remodelagem do ROV. Todo o processo é explicado ao longo do texto, o qual abrange desde a modelagem matemática até a programação e testes físicos.

As principais motivações deste trabalho são: aplicar os conhecimentos adquiridos durante o curso de engenharia na Poli USP como, por exemplo, eletrônica embarcada, mecânica dos fluidos, modelagem de sistemas e técnicas de controle; trabalhar com um domínio interessante e atual da robótica no Brasil, pois esse é um domínio científico recente, principalmente em relação aos aspectos experimentais.

A ideia final do projeto é desenvolver o ROV de forma a ter três graus de liberdade ativamente controlados. Além disso, funcionar com o modo de operação de controle manual.

Palavras-chave: ROV. Mecânica dos fluidos. Técnicas de Controle.

ABSTRACT

This part of the work consists in the modeling and hydrodinamical analysis of our ROV, project of controllers that take our ROV to the desired state, use of electronics capable of making the ROV moveable and controllable, replacement of all electronics from the beggining, motors, propellers, sensors, protection between control electronics and motor, a joystick development and the ROV remodeling. The entire process is explained along the text, with ranges from mathematical modeling to programming and physical tests.

The goal motivations of this work are to apply the knowledge acquired during the engineering course at Poli USP, such as embedded electronics, fluid mechanics, systems modeling and control techniques; work with an interesting and current field of robotics in Brazil, because this is a new scientific field, especially regarding experimental aspects.

The final idea of this project is to implement an ROV in order to have three degrees of freedom actively controlled. In addition, a manual operation mode of control.

Keywords: ROV. Fluid mechanics. Control Techniques.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ROV *Remotely Operated Vehicle*

NDF Núcleo de Dinâmica dos Fluidos

USP Universidade de São Paulo

SUMÁRIO

1 INTRODUÇÃO

Os oceanos são uma fonte muito rica de recursos para a humanidade e grande parte dessa riqueza é ainda inexplorada. Dessa forma, surge um grande desafio para a engenharia: desenvolver e implementar novas tecnologias que viabilizem o desbravamento das profundezas dos oceanos. Vale ressaltar a necessidade de explorar esses recursos naturais de maneira sustentável e para isso, conhecer o ambiente marinho será muito importante.

Os veículos subaquáticos operados remotamente são uma opção muito interessante como ferramenta de pesquisa, pois podem coletar dados, permitir observações, realizar monitoramentos, efetuar trabalhos de implantação e manutenção de instalações subaquáticas a grandes profundidades e com a grande vantagem de não colocar vidas humanas em risco. Mesmo que esses veículos já sejam bem utilizados na indústria de extração de petróleo, no Brasil, existem poucos trabalhos concernentes a pesquisas experimentais que abordam a construção física desses equipamentos.

2 OBJETIVOS

O projeto tem como objetivos gerais a implementação de um veículo subaquático nos três graus de liberdade: translação no eixo vertical, translação no eixo horizontal e rotação em torno do eixo vertical.

Durante a atual etapa do projeto, os objetivos são:

- Finalizar a remodelagem do ROV
- Resolver e implementar o problema de rastreamento
- Finalizar a montagem do ROV
- Integração do ROV

3 FASE INICIAL DO PROJETO NO 2º SEMESTRE DE 2015

Este projeto foi iniciado no segundo semestre de 2013 e primeiro semestre de 2014 pelo Diego Antônio Moreira, sendo que o atual ROV foi pensado para ser um submarino acadêmico segunda versão de protótipo da Engenharia Elétrica, ênfase em Automação e Controle (LAC). O projeto foi continuado pelo Fernando dos Santos Barbosa no primeiro semestre de 2014 juntamente com o Diego e no segundo semestre de 2014 com o Ricardo Hitoshi Nakano. No primeiro semestre de 2015 o projeto prosseguiu com Ricardo e Leandro Caboatan dos Santos.

Além disso, o projeto contou com a ajuda de dois alunos franceses que cursavam engenharia naval e faziam diploma duplo na Poli: Thierry Malavaud e Gaspard Douxchamps. Eles fizeram seu trabalho de formatura estudando os parâmetros hidrodinâmicos de um ROV, com base no nosso protótipo.

No segundo semestre de 2015 o projeto prosseguiu com Leandro e Sabrina Nabuco de Oliveira. Naquele momento o projeto passava por uma reestruturação física com a troca de motores, hélices, e plataforma de programação. Assim, era necessária a remontagem e remodelagem do ROV vem como a consequente revisão dos parâmetros utilizados em toda sua programação já existente.

Nessa etapa de desenvolvimento do ROV foram propostas a troca de motores e hélices, o uso do filtro de Kalman para melhorar o sensoriamento e a aplicação de controle multivariável para finalmente programar o controle horizontal (resolver o problema de rastreamento) que até essa etapa não havia sido desenvolvido.

4 CRONOGRAMA

Os cronogramas utilizados para o segundo semestre de 2015 e primeiro semestre de 2016 são mostrados a seguir.

Atividades	Subatividades	Julho				Agosto				Setembro				Outubro				Novembro			
		S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4
Troca de motores	Delineamento do requisitos																				
	Pesquisa																				
	Compra																				
Troca de hélices	Delineamento do requisitos																				
	Pesquisa																				
	Compra																				
Controle multivariável	Estudo e desenvolvimento																				
	Remodelagem do sistema																				
	Discretização																				
	programação																				
	testes																				
Filtro de Kalman	Estudo dos sensores																				
	Estudo e desenvolvimento																				
	Aplicação																				
	testes																				
integração	programação																				
	testes																				
Apresentações	Apresentação 1																				
	Apresentação 2																				
	Apresentação 3																				

Figura 1 - Cronograma 2º semestre de 2015

Atividades	Subatividades	Fevereiro				Março				Abril				Maio				Junho			
		S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4
Modelagem	Reparos remanescentes na modelagem																				
	Novo cálculo dos parâmetros																				
	Novo cálculo das plantas																				
Controle Automático	Resolver o problema de rastreamento																				
	Simulações																				
Estruturação do ROV	Aquisição das peças remanescentes																				
	Resolução dos problemas físicos de 2015																				
	Finalização da montagem do ROV																				
Programação	Na Raspberry Pi em python																				
Integração	Integração com a estrutura do ROV																				
	Teste Final																				
Apresentações	Apresentação 1																				
	Apresentação 2																				
	Apresentação 3																				

Figura 2 - Cronograma

5 ESTRUTURAÇÃO DO ROV

5.1 PLACA DE AQUISIÇÂO – RASPBERRY PI

No início do semestre de 2015 o ROV passou por uma reestruturação e aquisição de novos componentes. Esse capítulo será dedicado à exposição dos componentes e suas respectivas características.

A Raspberry Pi B+ foi adquirida com o objetivo de se ter uma saída na placa que não limite em relação ao tamanho dos fios utilizados. Além disso, a placa possibilita a utilização de Ethernet como saída, viabilizando a existência de uma interface homem-máquina.

O principal desafio da aquisição de uma nova placa tanto para o grupo anterior quanto para o grupo imerso na nova fase é a necessidade de refazer um circuito que já estava pronto, além disso faz-se necessário o aprendizado de um novo sistema operacional (Linux) e do estudo de como programar nele. A linguagem a ser utilizada é a Python, uma linguagem de programação de alto nível.

Figura 3 - Nova placa de aquisição - Raspberry pi B+

5.2 SENSORES

5.2.1 Câmera

Foi adquirida uma câmera com o objetivo de mapear a região desejada, para poder gravar e registrar o que há nas regiões onde o ROV passar. Além disso, a

câmera serve para ajudar no controle de direção e nos da uma visão melhor de onde o ROV esta se locomovendo.

Figura 4 - Camera - Raspberry pi B+

5.1.1 Sensor GY-80

O sensor GY-80, que contém os módulos giroscópio (L3G4200D), 3 eixos do Acelerômetro (ADXL345), 3 eixos Magnetômetro (HMC5883L) e mais o sensor de pressão e temperatura (BMP085), foi adquirido com a finalidade de implementar o controle de direção e monitorar de forma precisa o ROV.

Figura 5 - Sensor GY-80

5.3 MOTORES E ESC

Para o melhor desempenho do ROV, foram adquiridos novos motores, mais potentes e mais uniformes, possibilitando um controle mais confiável.

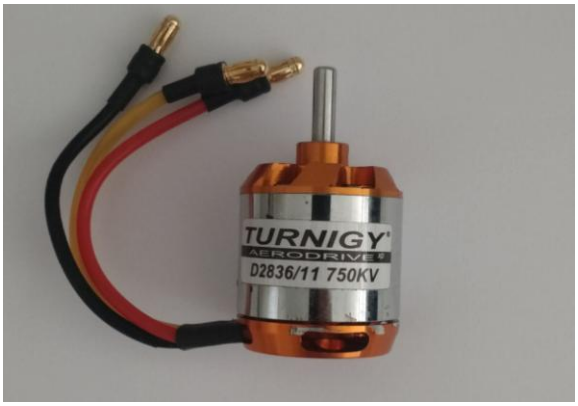

Figura 6 – Motor com eixo de 4 mm

Figura 7 – Motor com eixo de 3 mm

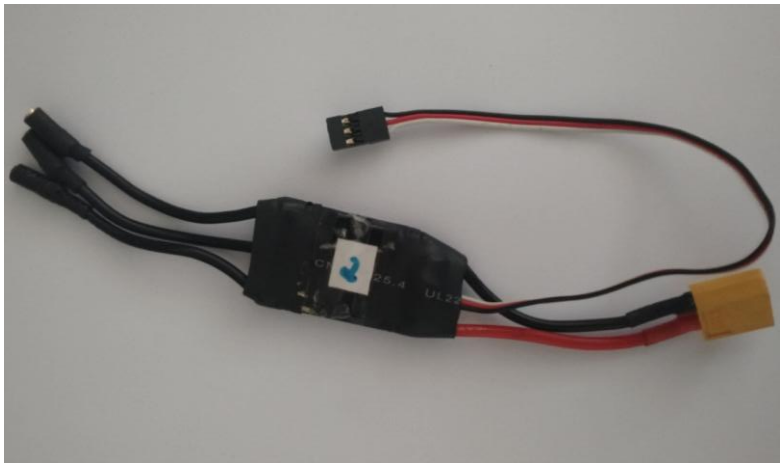

Figura 8 - ESC

Os motores são do tipo Brushless, ou seja, não possuem contatos internos, e são Outrunner, ou seja, a parte que gira é a parte externa do motor, não são passíveis de inversão de corrente portanto não invertem o sentido da rotação, absorvem mais corrente que os motores escovados e necessitam para um correto funcionamento serem ligados à ESC`s (do inglês Electronic Speed Controller, traduzido para controlador eletrônico de velocidade). Por serem controlados por ESC`s individuais esses motores são muito confiáveis em seu funcionamento e a particularidade fundamental deles é que o conjunto ESC + motor deve ser alimentado por uma tensão de aproximadamente 12V e controlado por PWM (do inglês Pulse Width Modulation, traduzido para modulação de largura de pulso).

A corrente a mais que é puxada pelo conjunto ESC + motor provem da fonte de alimentação e é controlada pela ESC, e suporta um pico de 30 A em funcionamento normal de acordo com especificações, não ocasionando riscos de queima para o circuito de controle, a Raspberry Pi, que é relativamente sensível.

Temos uma *ESC* para cada motor e essas servem de maneira que a corrente alta que o motor exige para seu funcionamento não prejudique o restante do circuito. A *ESC* funciona ao mesmo tempo como uma interface de comunicação entre o motor e o circuito de controle e como uma proteção. Essa proteção é o motivo pelo o qual utilizávamos anteriormente o Shield de proteção do Arduino, o qual não se mostrou muito eficiente. A proteção realizada pela ESC é realizada de forma que o motor e o circuito de controle não tenham nenhum compartilhamento de corrente, de forma que a corrente que o motor e o circuito de controle utilizam sejam completamente independentes. O modo pelo o qual o circuito de controle e a ESC se comunicam é a partir do sinal PWM.

5.3.1 Recipientes para os motores

Devido ao nosso robô precisar funcionar sobre a água e esses motores não poderem ter contato com a água, realizamos o projeto de quatro recipientes, as quais deverão isolar cada motor da água.

Segue, nas figuras 21 a 23, o desenho do modelo de caixinha e das tampas pretendido incialmente.

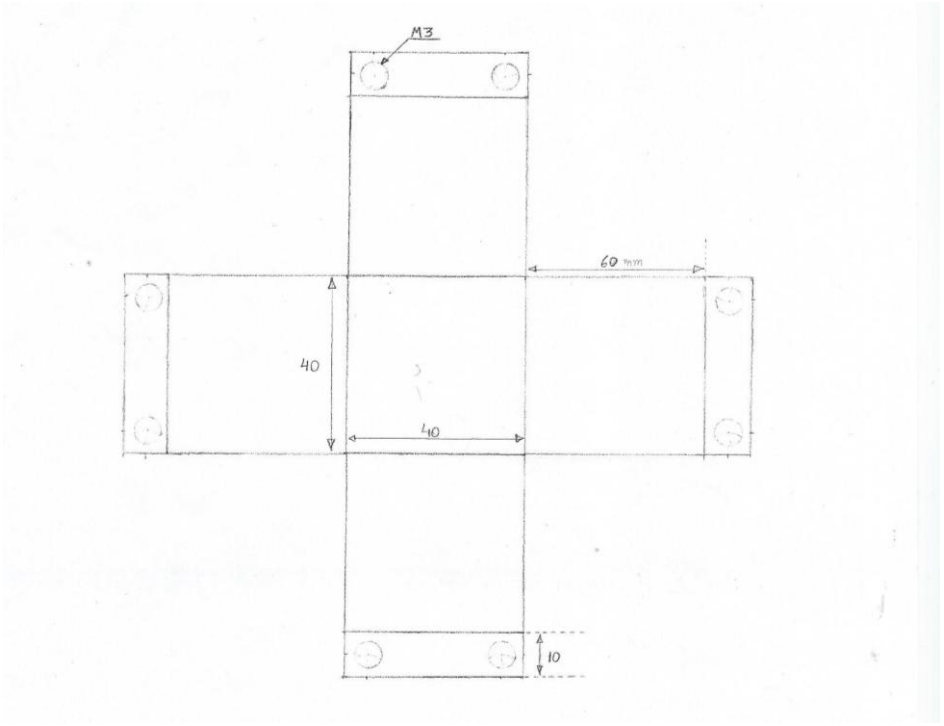

Figura 9 – Desenho da caixinha para os motores

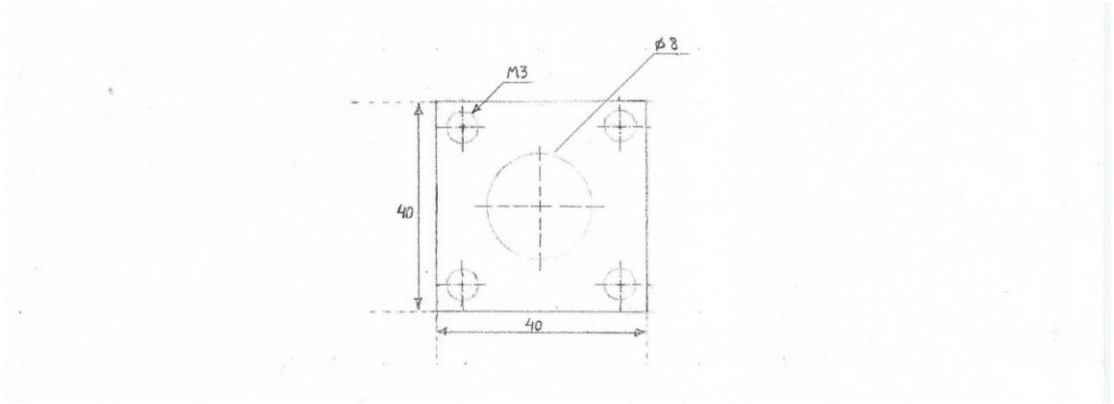

Figura 10 – Tampa para o motor com eixo de 3 milímetros

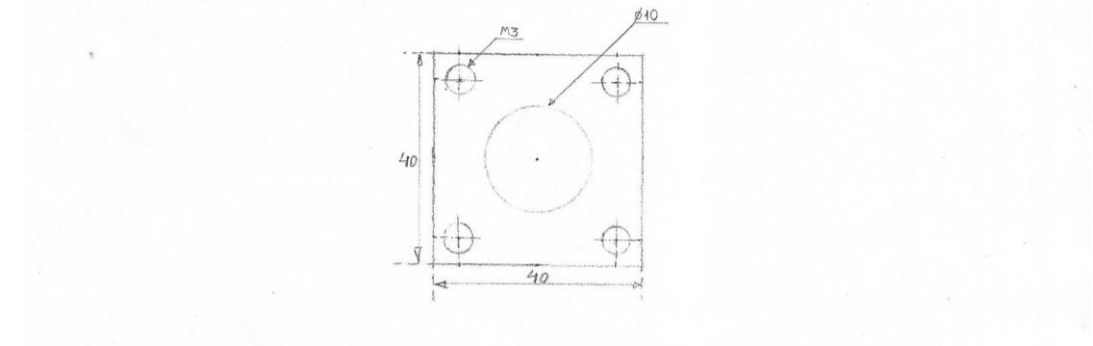

Figura 11 - Tampa para o motor com eixo de 4 milímetros

Observa-se pelas figuras que a base das quatro caixinhas é a mesma, porém há diferença nas tampas.

A base se trata de uma peça que seria fabricada a partir de uma chapa de alumínio e que tem suas laterais a partir do dobramento das extensões da base. Além das laterais da caixa, nessa peça temos também que a partir de cada lateral tem-se uma pequena extensão para possibilitar a passagem de parafusos para a fixação da tampa com a base.

Após os devidos dobramentos da peça base, ainda seria necessário que uníssimos de alguma maneira uma lateral a outra e também que essa união sirva de isolamento à água, objetivo principal da caixinha.

A solução encontrada para a união das laterais foi a solda. A partir dessa garantir-se-ia união sólida e resistente à água.

As tampas foram pensadas a partir de dois modelos, os quais se caracterizam para se adequarem aos dois tipos de motores disponíveis. Ambas as tampas possuem dimensões compatíveis com a base para o encaixe e parafusar a tampa e com a base.

A diferença entre as tampas se encontra no rolamento que vai com cada uma delas. Esse rolamento tem tamanho dependente do tamanho do eixo dos motores, que possuem diâmetros de 3 e 4 mm, fazendo necessário que o diâmetro interno dos rolamentos sejam compatíveis.

Os rolamentos são conectados às tampas por interferência para garantirem isolamento dessas à água.

A necessidade de termos presentes rolamentos às nossos recipientes se encontra na eliminação da dificuldade de rotação dos motores, dificuldade que era observada com a disposição dos motores antigos e suas caixinhas que dificultavam sua rotação. O rolamento é o item mecânico desenvolvido especificamente para isso.

Figura 12 – Rolamentos para os motores

Figura 13 – Hélices com 2 pás

Figura 14 – Hélices com 3 pás

A eliminação da dificuldade de rotação se torna necessária devido a um motivo simples, quanto maior a dificuldade de rotação para o motor mais corrente esse exige da fonte de alimentação para compensar a dificuldade. Esse, aliás, é um motivo muito provável dos defeitos ocorridos nos testes na água de 2014.

Como sabemos que os itens elétricos, fora os motores e seu controlador, funcionam com correntes na ordem de mili ampères, precisamos eliminar a possibilidade de acontecer algo que os queime.

O fabricante dos motores especifica que a corrente de pico máxima é de 20 A e as ESC`s que possuímos estão especificadas a aguentar até 30 A.

Na tentativa de fabricar os recipientes para os motores nos deparamos com diversos problemas, os quais devido a não termos experiência de fabricação e de dependermos do serviço pedido para a oficina e técnicos do prédio da engenharia mecatrônica da Escola Politécnica.

Outro problema ainda foi o acesso aos tanques de provas presentes na Escola Politécnica, sendo eles o TPN (Tanque de Provas Numérico) e os dois tanques presentes no prédio da mecatrônica, na parte de mecânica dos fluidos, o qual possui acesso ao software AQWA e que é necessário para revermos a modelagem.

A falta de experiência com fabricação provém da inexistência de disciplinas para tal em nossa grade curricular da engenharia elétrica. O problema de dependência para com os técnicos gera uma série de problemas, dentre os quais problemas burocráticos, falta de noções de processo de fabricação e inexperiência com desenhos profissionais para que um profissional que fabrica peças possa realizar o trabalho adequadamente.

Tais problemas para a fabricação dos recipientes geraram demora para que os desenhos chegassem aos técnicos, problemas de comunicação com esses profissionais e gerou a necessidade de mudarmos completamente o projeto dos recipientes pela inadequação da ideia de como seria concebida os recipientes e tampas para com o modo de fabricação, que foi pensado ser simples porém se mostrou um empecilho.

Após passarmos por tais problemas e de conversas com os técnicos expondo nossas ideias e ouvindo a opinião deles quanto aos problemas de fabricação e pontos a melhorar no projeto acabamos por realizar o projeto de outro tipo de caixinha para o isolamento dos motores.

5.3.2 Novo projeto de recipientes para os motores

Para o novo projeto dos recipientes e tampas reformulamos o formato físico dos recipientes que conterão os motores e acrescentamos detalhes para garantir a fixação efetiva dos motores e rolamentos.

Optamos pelo formato cilíndrico para podermos efetuar o fechamento do recipiente através de rosqueamento. Essa técnica é mais eficiente pois além de ser um processo acessível para usinar, eliminamos eventuais problemas de vazamento que poderiam surgir através dos parafusos como elaborado no primeiro projeto.

Como os motores adquiridos possuem rotação externa (*outrunner*), a necessidade de evitar que sua superfície de rotação encoste na superfície dos recipientes, evitando assim o atrito entre eles e consequente perda de rotação e

torque. Dessa forma fixaremos os motores pelas bases e calculamos os diâmetros (30mm) dos recipientes de forma a garantir uma distância segura entre as superfícies.

Um dos motores apresenta uma saliência em sua base que também rotaciona externamente, para evitar atrito com essa região quando fixarmos a base, acrescentamos uma depressão central.

No novo projeto acrescentamos um orifício para a passagem dos fios trifásicos dos motores. O diâmetro do orifício foi calculado para ser o suficiente apenas para a passagem dos fios (8mm) e depois será fechado com shrink tube (plástico termo retrátil) para evitar vazamento de água para o interior do recipiente.

O projeto da tampa também foi reelaborado, além da nova forma de fechamento através de rosqueamento, acrescentamos apoios para a fixação dos rolamentos evitando inesperadas instabilidades.

Finalmente, mantivemos o alumínio como material de fabricação para evitar oxidação já que o recipiente estará em contato direto com a água. Inicialmente optamos por uma espessura de 1mm para facilitar as trocas de calor, porém, novamente ao conversar com colegas da engenharia mecânica fomos informados que para o rosqueamento ser possível há padrões de espessura a serem seguidos e por isso devemos aumentar a espessura do recipiente para que o processo de usinagem da rosca não fure o material. As demais características do recipiente e tampas permanecem as mesmas.

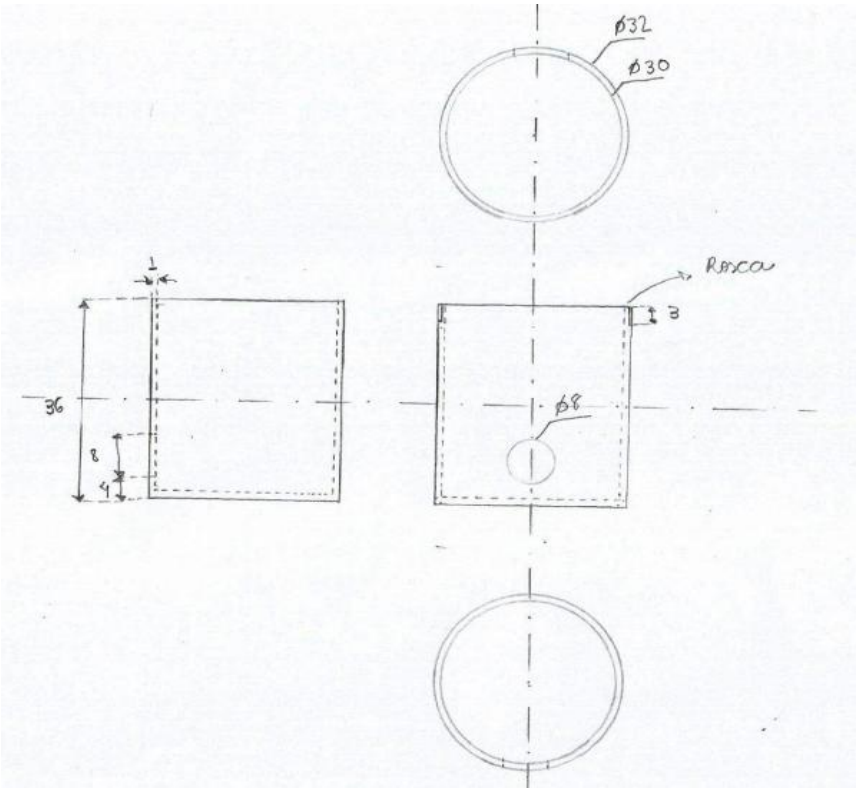

Figura 15 – Representação gráfica do recipiente para o motor com eixo de 3 milímetros

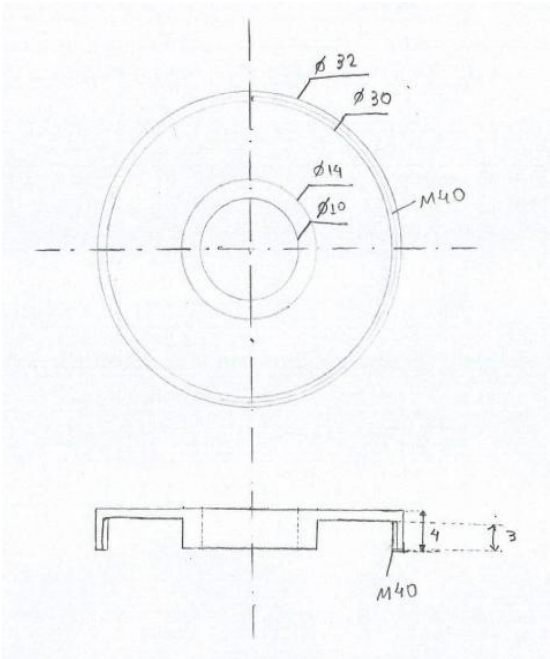

Figura 16 – Representação gráfica da tampa para o motor com eixo de 3 milímetros

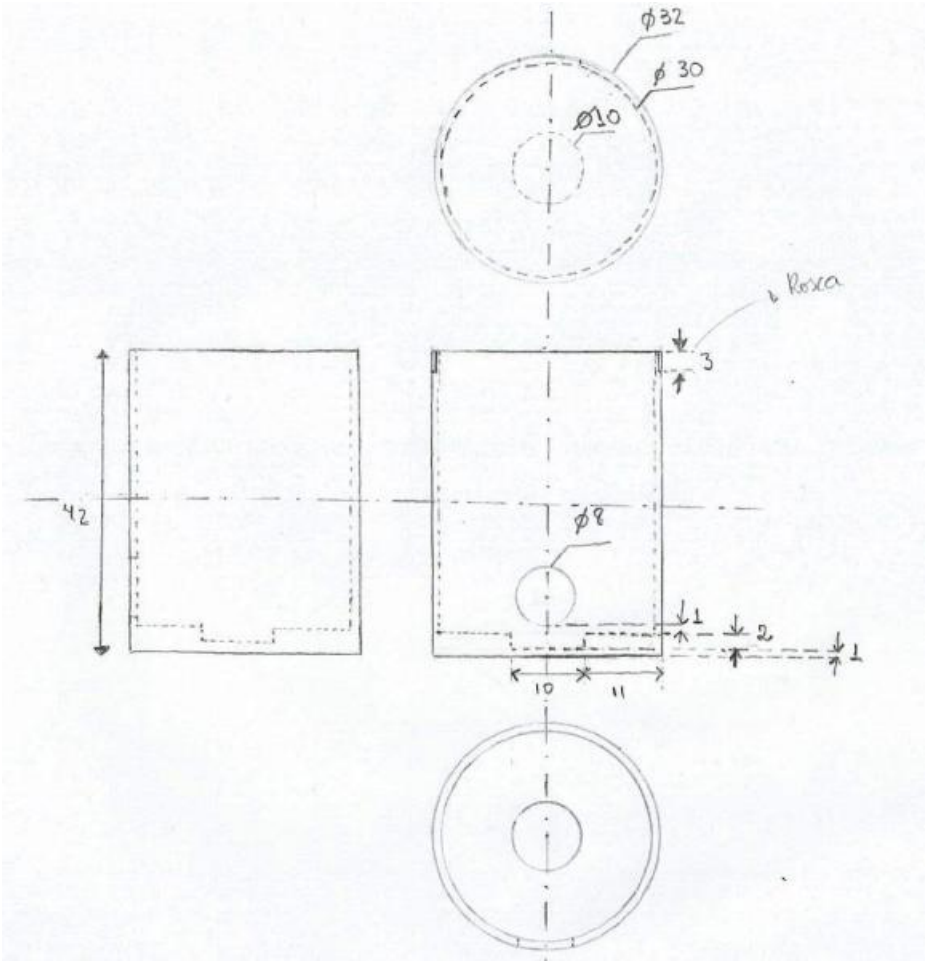

Figura 17 - Representação gráfica do recipiente para o motor com eixo de 4 milímetros

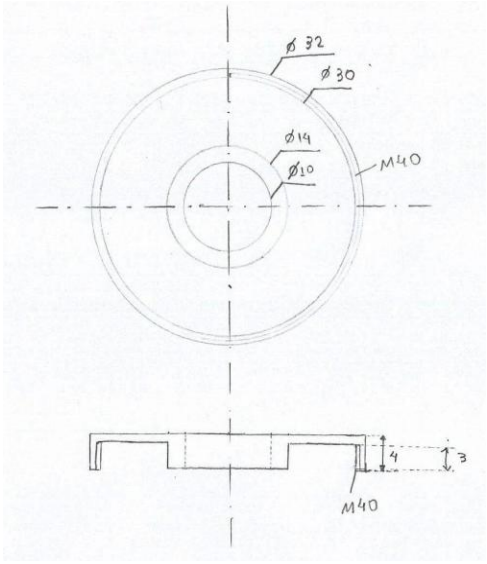

Figura 18 - Representação gráfica da tampa para o motor com eixo de 3 milímetros

Nessa etapa do projeto tivemos um atraso no cronograma uma vez que a usinagem dos recipiente dos motores apresentou diversos problemas. Por sermos alunos da Engenharia Elétrica, em nossa grade curricular não existem disciplinas que nos ensinem os processos de usinagem de uma peça. Deste modo, terceirizamos o trabalho para a oficina de mecatrônica. Entretanto a oficina recebe diariamente pedidos de diversos departamentos, incluindo de professores da graduação que tinham prioridade sobre o nosso projeto. O prazo de dez dias estipulado pela própria oficina durou um mês e meio, e as justificativas apontadas para nós foram projetos prioritários, problemas com a falta de ferramentas (alargador) que tivemos que fornecer para a oficina, e com a ausência de ferramenta precisa para realizar a rosca que fecharia o recipiente. O trabalho foi então enviado para uma oficina do PME quando finalmente foi concluída.

As imagens a seguir mostram o processo de fabricação dos recipientes dos motores em andamento. E em seguida temos os recipientes dos motores concluídos.

Figura 19 – Tampas dos Recipientes

Figura 20 – Processo de Fabricação dos Recipientes

Figura 21 – Recipientes usinados

Com a finalização da usinagem dos recipientes dos motores, foi possível realizar a pesagem dos recipiente juntamente com os motores e os rolamentos. Dessa forma, foi possível dar continuação ao cálculo das novas matrizes inerciais que permitem rever a modelagem do nosso ROV. Inserimos a matriz de centro de massa no programa de simulação AQWA.

Os novos pesos obtidos foram 111g para os motores em laranja que serão usados na direção vertical e 0,85g para os motores em vermelho que serão usados na direção horizontal. O critério utilizado para a escolha de quais motores usar em

cada posição foi a potência de cada motor. Acreditamos que o motor de maior potência deva ser utilizado na direção vertical, já que será responsável pelo deslocamento nessa direção que exige mais força, já que pode atuar contra a ação da gravidade.

6 REMODELAGEM DO ROV

A troca de componentes do ROV para o segundo semestre de 2015 acarretou na necessidade de remodelagem do ROV. O ROV sofreu uma reestruturação física e consequentemente se modelo matemático também precisou ser alterado.

Atualmente, o ROV possui quatro motores de corrente contínua, tratam-se de motores *outrunner*, ou seja, a sua rotação é externa. Dois dos motores estão localizados na parte superior, disposto verticalmente e dois na parte traseira, disposto horizontalmente. Os dois motores traseiros conferem ao modelo seu deslocamento no plano horizontal, enquanto que os superiores são usados para controlar sua profundidade na água.

O controle atual é feito por uma Raspberry Pi e a linguagem de programação utilizada é a Python. Com a Raspberry é possível realizar o controle dos motores passando pelas ESCs, o controle dos motores é feito através de pulsos PWM fornecidos pela Raspberry. A Raspberry também é responsável por controlar as luzes e sensores.

6.1 DESCRIÇÃO MATEMÁTICA DA PLANTA

Uma vez que a modelagem do ROV já foi realizada anteriormente nesse projeto, e a alteração de componentes não afeta tal modelagem, não despenderemos tempo reexplicando a forma como a descrição matemática da planta foi obtida, tal explicação pode ser vista em [1].

Das simplificações obtidas em [1], temos:

$$\upsilon = [u \quad v \quad w \quad p \quad q \quad r]^T$$

$$\upsilon = [u \quad 0 \quad w \quad 0 \quad 0 \quad r]^T$$

Onde υ é o vetor de velocidades no sistema solidário ao ROV. Os três primeiros componentes fazem referência à velocidade linear em avanço, deriva e afundamento, respectivamente; já os três últimos são referentes às velocidades angulares em jogo, arfagem e guinada, respectivamente.

$$\eta = [x \quad y \quad z \quad \varphi \quad \theta \quad \psi]^T$$

$$\eta = [x \quad y \quad z \quad 0 \quad 0 \quad \psi]^T$$

Onde η é o vetor referente ao sistema fixo na embarcação. Os três primeiros componentes descrevem a posição cartesiana do ROV, enquanto que os três últimos, os respectivos ângulos.

E o modelo hidrodinâmico do ROV pode ser resumido a:

$$M\dot{\upsilon} + D(\upsilon)\upsilon = \tau$$

- τ é o vetor de forças de controle produzidas pelos motores.

Ainda é possível fazer as seguintes simplificações [2]:

- Como não existe nenhum motor na direção de deriva, é admitido que a velocidade neste eixo seja sempre nula, ou seja, $v = 0$.
- Devido à estabilidade em relação ao plano horizontal, admite-se $p = q = 0$. Não há velocidade angular em nenhuma destas direções.
- Como implicação da simplificação acima, $\varphi = \theta = 0$.

Como mencionado nas considerações acima, o sistema resultante é desacoplado em relação aos movimentos horizontal e vertical e, assim, pode ser dividido em dois subsistemas independentes [3] onde:

- u, r, x, y, ψ correspondem ao movimento planar horizontal;

- z, w correspondem ao movimento vertical.

Reescrevendo o modelo hidrodinâmico separadamente para cada subsistema, temos:

- Modelo horizontal:

$$\begin{bmatrix}\dot{u}\\ \dot{r}\end{bmatrix} = \begin{bmatrix}\frac{X_u}{m - X_{\dot{u}}} & 0\\ 0 & \frac{N_r}{I_{zz} - N_{\dot{r}}}\end{bmatrix}\begin{bmatrix}u\\ r\end{bmatrix} + \begin{bmatrix}\frac{1}{m - X_{\dot{u}}} & \frac{1}{m - X_{\dot{u}}}\\ \frac{-D_{1y}}{I_{zz} - N_{\dot{r}}} & \frac{-D_{2y}}{I_{zz} - N_{\dot{r}}}\end{bmatrix}\begin{bmatrix}F_1\\ F_2\end{bmatrix}$$

- Modelo Vertical:

$$\dot{w} = \left(\frac{Z_w}{m - Z_{\dot{w}}}\right)w + \left(\frac{1}{m - Z_{\dot{w}}}\right)F_V$$

Com $F_V = F_3 + F_4$.

6.2 REMODELAGEM DOS PARÂMETROS HIDRODINÂMICOS

Durante a remodelagem, os parâmetros hidrodinâmicos foram obtidos por meio de simulação computacional.

6.2.1 Inércia Estrutural

A princípio fez-se necessária a estimativa da inércia estrutural, cálculo realizado analiticamente.

O veículo foi dividido em oito peças consideradas homogêneas. As dimensões de cada peça foram estimadas e são apresentadas na tabela a seguir (em mm):

Comprimento	L	343
Comprimento do tubo principal	L_2	318
Comprimento dos tubos secundários	L_3	305
Boca	B	335
Largura entre centros dos tubos secundários	B	207
Diâmetro do tubo principal	D_1	110 (105, 120)
Diâmetro do tubo secundário	D_2	40 (40,3)
Altura	H	198
Altura no topo do tubo principal	H	180
Espaçamento entre armaduras de proa e popa	L	125
Comprimento das armaduras longitudinais	$L_{a,l}$	225
Comprimento das armaduras transversais	$L_{a,t}$	180
Comprimento das armaduras oblíquas	$L_{a,o}$	85
Largura das armaduras metálicas	l_a	25
Espessura das armaduras metálicas	e_a	3
Comprimento dos compartimentos dos motores I	L_{m1}	46
Diâmetro dos compartimento dos motores I	D_{m1}	36
Comprimento dos compartimentos dos motores II	L_{m2}	46
Diâmetro dos compartimento dos motores II	D_{m2}	36
Diâmetro dos hélices	D_h	54
Comprimento dos eixos dos hélices	L_h	34
Diâmetro do sensor de pressão	D_p	45
Comprimento do sensor de pressão	L_p	35
Diâmetro das lâmpadas	D_l	28
Comprimento das lâmpadas	L_l	39
Diâmetro do umbilical de alimentação geral	$D_{c,1}$	6
Diâmetro dos umbilicais de alimentação auxiliares	$D_{c,2}$	4
Diâmetro dos umbilicais de comunicação	$D_{c,3}$	3
Comprimento dos umbilicais	L_c	5800

Tabela 1 - Dimensões do ROV

Com esses valores e o peso medido do ROV (3,524 kg), obteve-se a matriz de inércia estrutural, e as coordenadas dos centros de flutuação e de massa, mostrados a seguir:

$$M = \begin{bmatrix} 3{,}524 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3{,}524 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3{,}524 & 0 & 0 & 0 \\ 0 & 0 & 0 & 46302 & 0 & 787 \\ 0 & 0 & 0 & 0 & 35885 & 0 \\ 0 & 0 & 0 & 787 & 0 & 62105 \end{bmatrix}$$

	x (mm)	y (mm)	z (mm)
Centro de massa (G)	-6	0	-48
Centro de carena (B)	-11	0	-32

Tabela 2 - Centro de Massa e Flutuação

Modificou-se então os modelos computacionais para a simulação. Novamente utilizamos o software Aqwa® para a o desenho do modelo 3D e determinação da Matrizes de Massa Adicional. Apesar de boa parte do modelo já estar construído nos arquivos elaborados em etapas anteriores, foi necessário um tempo de aprendizado e reconhecimento do ambiente do software utilizado. Também foi necessária uma releitura minuciosa da forma como a modelagem foi feira no princípio do projeto.

As figuras a seguir mostram o modelo antigo e o modelo atual do ROV.

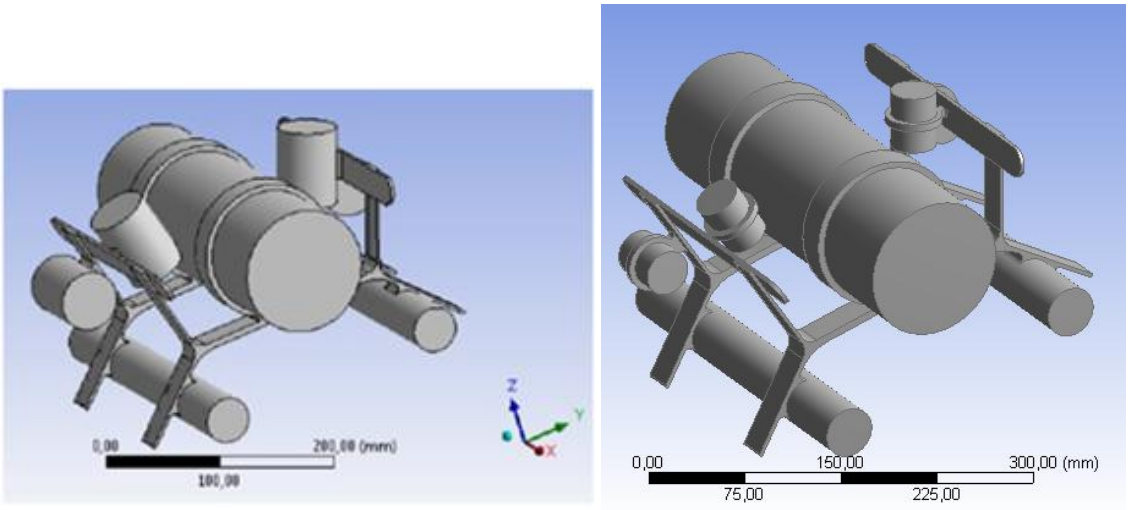

Figura 22 - Modelo 3D Antigo do ROV (esquerda) e modelo 3D Atual do ROV (direita)

6.2.2 Inércia Adicional

Prosseguindo o trabalho com o software Aqwa®, vários valores de inércia adicional foram obtidos em função da frequência de oscilação imposta. Sendo que as variações eram pequenas menos de 1 %, fez-se a média dos valores para se

obter a matriz de inércia adicional simulada, desconsiderando os termos desprezíveis chegou-se a (unidades métricas em milímetros):

$$\rho V M_A = \begin{bmatrix} 1{,}1 & 0 & 0 & 0 & 37{,}7 & 0 \\ 0 & 3{,}4 & 0 & -75{,}5 & 0 & 21{,}3 \\ 0 & 0 & 4 & 0 & -19{,}5 & 0 \\ 0 & -75{,}8 & 0 & 21591 & 0 & 0 \\ 37{,}7 & 0 & -19{,}6 & 0 & 25052 & 0 \\ 0 & 21{,}2 & 0 & 0 & 0 & 25060 \end{bmatrix}$$

Em termo de M_A tem-se:

$$M_A = \begin{bmatrix} 0{,}3 & 0 & 0 & 0 & 10{,}7 & 0 \\ 0 & 1{,}1 & 0 & -21{,}5 & 0 & 6{,}1 \\ 0 & 0 & 1{,}2 & 0 & -5{,}6 & 0 \\ 0 & -21{,}5 & 0 & 6134 & 0 & 0 \\ 10{,}7 & 0 & -5{,}6 & 0 & 7117 & 0 \\ 0 & 6 & 0 & 0 & 0 & 7119 \end{bmatrix}$$

7 MALHA DE CONTROLE

A partir das novas matrizes obtidas acima, bem como valores já estabelecidos em [1] é possível recalcular as plantas que serão utilizadas para a resolução do problema de rastreamento e de controlabilidade.

O problema do projeto é encontrar um compensador K(s) de forma que y acompanhe r com uma precisão pré-estabelecida e com um esforço de controle razoável, a despeito das presenças de perturbações d e do erro de medida n.

A seguir temos o diagrama de blocos do sistema nominal, em que:

K(s) - compensador a ser projetado.

G(s) - modelo nominal da planta a ser controlada

d(s) - representa os sinais de perturbações externas

n(s) - representa os erros de medida do sistema

r1(s) - sinal de referencia a ser seguido pela saída

r(s) - sinal de referencia após pré-filtro

e(s) - erro entre a referencia e a saída medida

u(s) - sinal de controle

y(s) - saída controlada do sistema

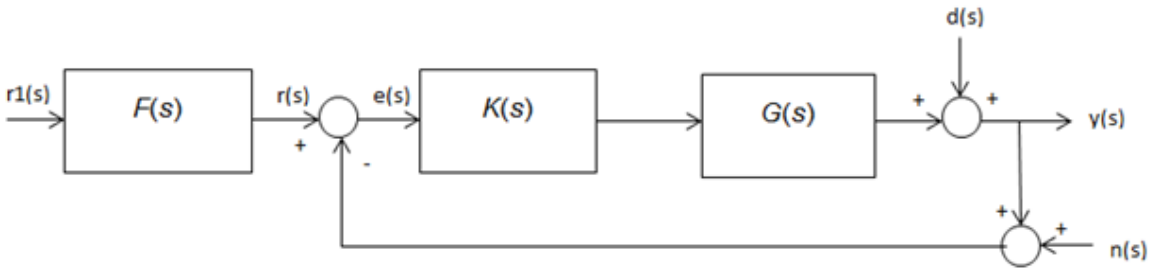

Figura 23 - Diagrama de Blocos Canônico

7.1 SIMULAÇÕES PLANTA VERTICAL

Com o intuito de entender melhor como a planta reage aos estímulos, um programa no MatLab foi desenvolvido para desenhar vários gráficos já conhecidos na área de controle. Para isso, duas plantas foram utilizadas:

$$G_1(s) = \frac{0.1321}{s + 0.1116} \; e \; G_2(s) = \frac{1}{s} * \frac{0.1321}{s + 0.1116}$$

Isso devido ao fato de que a primeira planta tem como saída a velocidade de afundamento do ROV e, como o intuito do projeto é controlar a profundidade, colocou-se um integrador na saída. Então, $G_1(s)$ tem força em Newton como entrada e velocidade em metros por segundo como saída, enquanto que $G_2(s)$ possui a mesma entrada, porém como saída a profundidade em metros.

A primeira figura a seguir mostra as respostas de ambos os modelos a uma entrada degrau de 1 N.

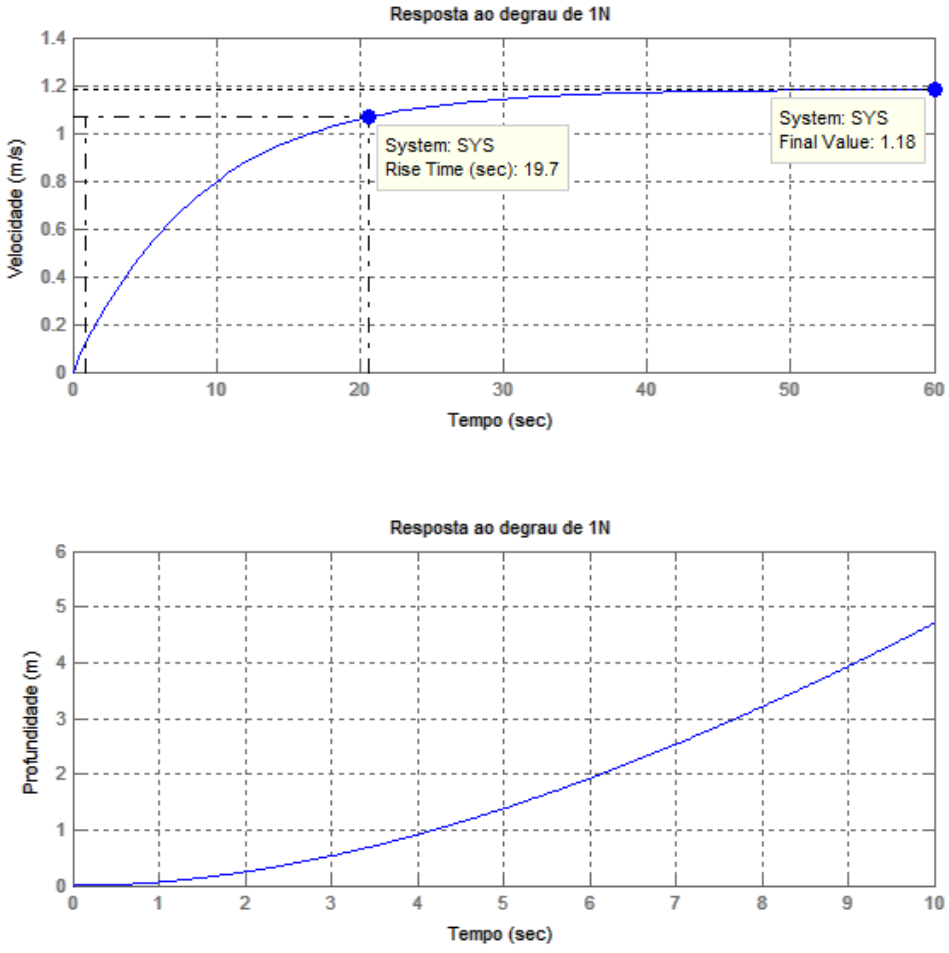

Figura 24 - Resposta ao Degrau

É possível verificar que, com esta entrada, a velocidade final de afundamento do ROV é de 1.18 m/s e que, em aproximadamente 10s, estaria a 5m de profundidade. A próxima imagem mostra o Lugar das Raízes de ambos.

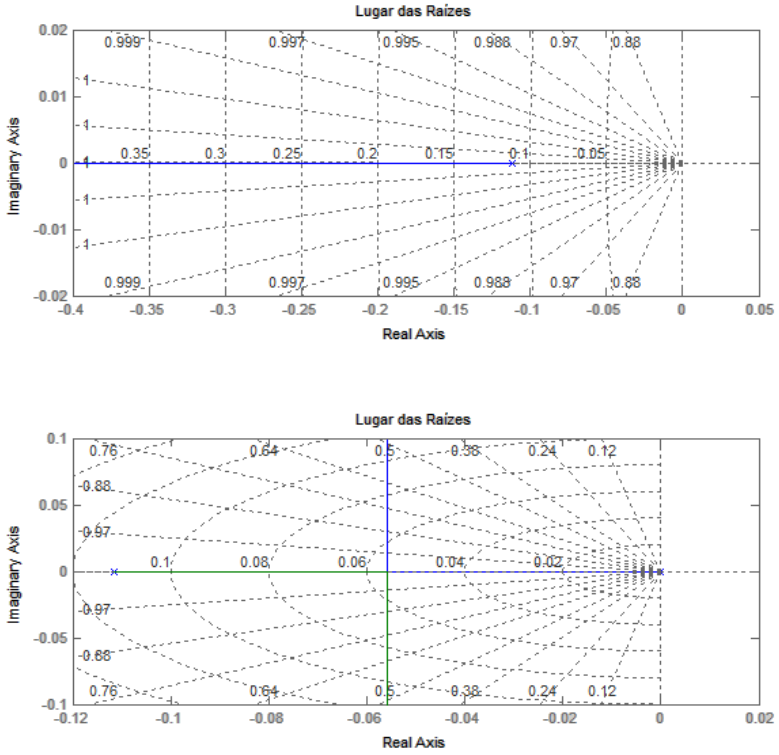

Figura 25 - Lugar Geral das Raízes

Esta imagem é bem esclarecedora quanto aos efeitos da implantação de um polo em zero na segunda planta. Pois bem, a próxima imagem representa o Diagrama de Nichols.

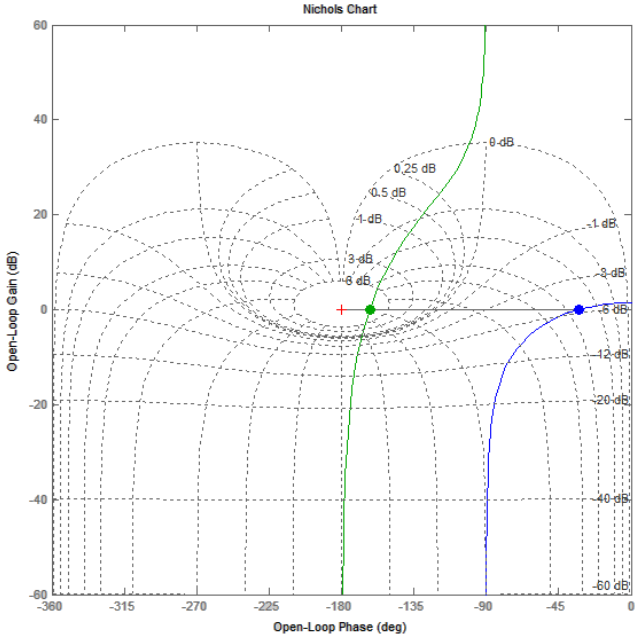

Figura 26 - Diagrama de Nichols

Nesta imagem decidiu-se colocar o resultado das duas plantas no mesmo gráfico, com o intuito de facilitar a comparação. O tracejado mais a direita é referente à primeira planta, com margem de fase de 148º e margem de ganho infinita. Já o segundo sistema possui uma margem de fase de 17.5º, também com margem de ganho infinita. Passemos então ao Diagrama de Bode.

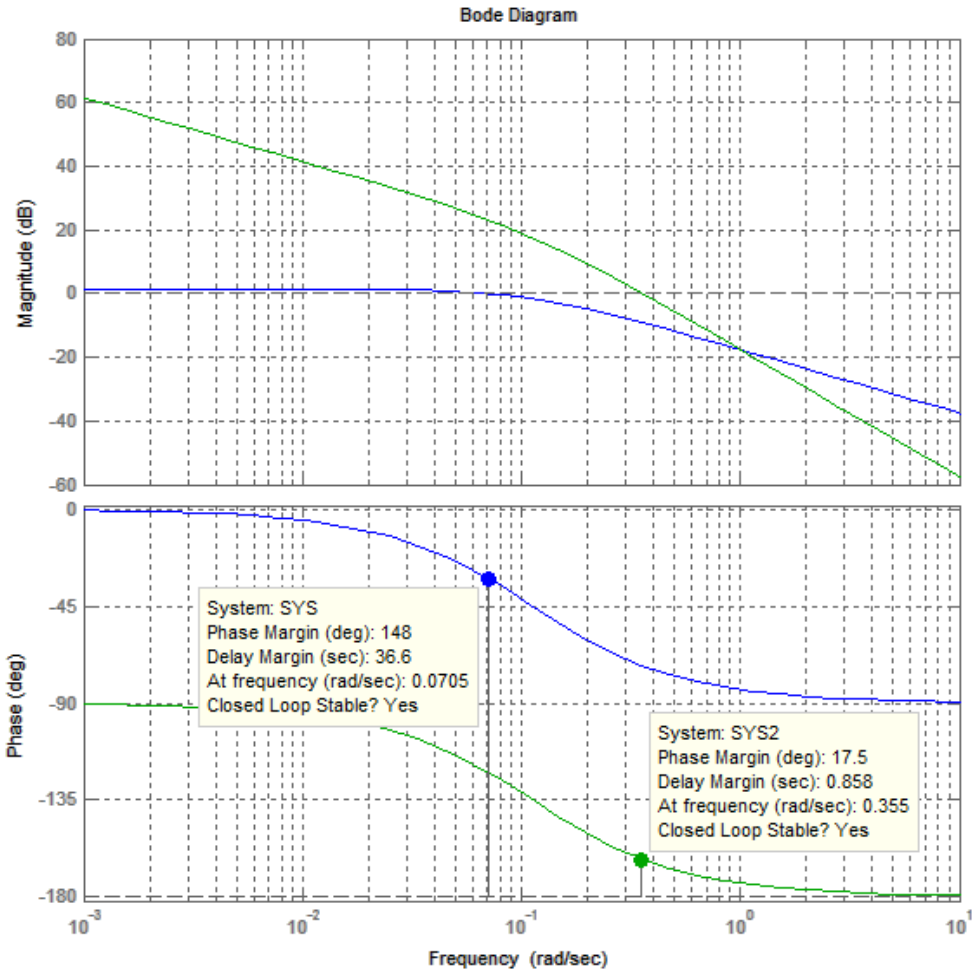

Figura 27 - Diagrama de Bode

Nesta imagem, a planta dois é a representada mais acima, em baixas frequências, no gráfico de módulo e mais abaixo no gráfico de fase. Este gráfico também é bem representativo sobre como a implantação de um polo é capaz de alterar a dinâmica da planta. O último diagrama a ser mostrado é o Diagrama de Nyquist.

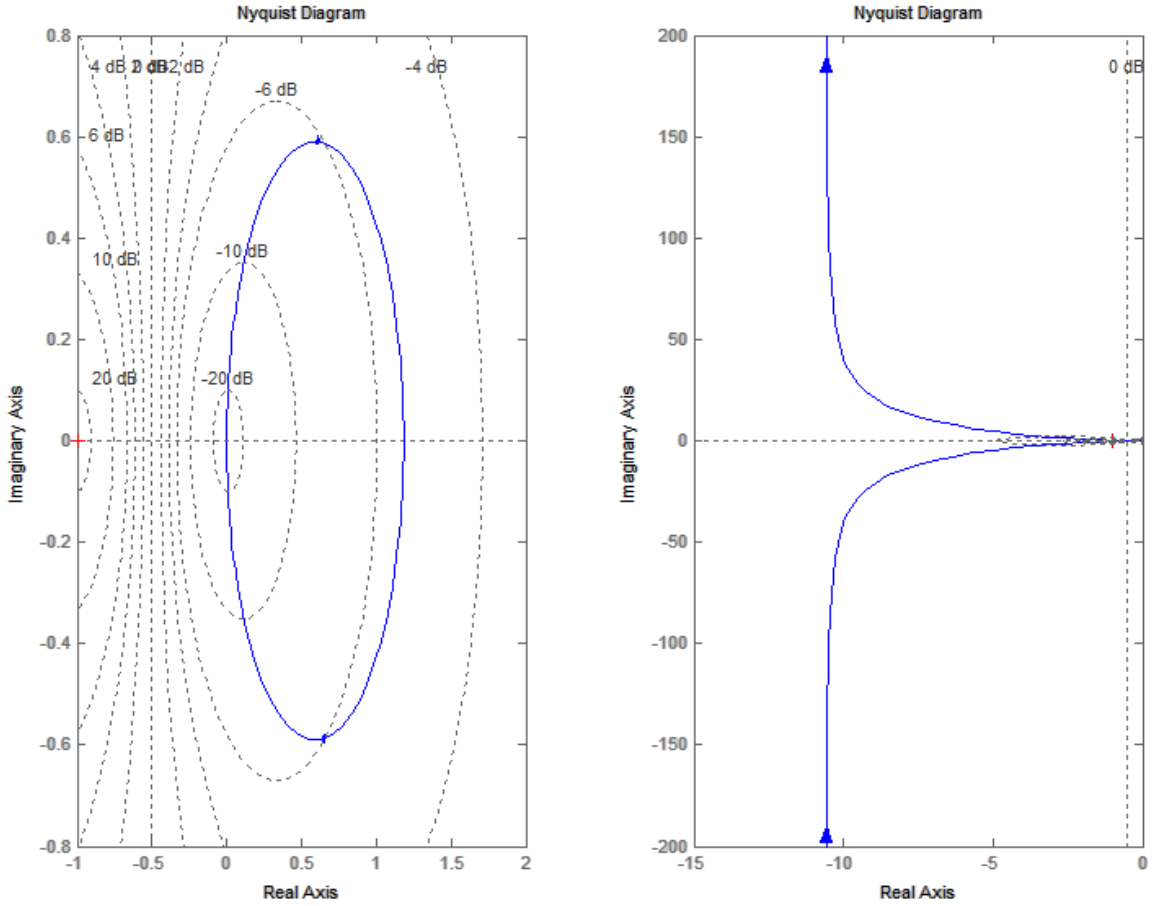

Figura 28 - Diagrama de Nyquist

O controlador para o modelo vertical foi realizado através de técnicas de QFT, baseado na teoria de controle robusto. Como não fazia parte do escopo do cronograma desse semestre e seria necessário um background teórico sobre o assunto ele não foi revisado nessa etapa. Passou-se então para a próxima etapa do cronogrâma, o controle horizontal.

7.2 ESPECIFICAÇÕES DA PLANTA HORIZONTAL

7.2.1 Desacoplamento

De acordo com o descrito em [1], houve uma necessidade de realizar o desacoplamento das entradas do modelo horizontal, que é descrito a seguir.

:

$$\dot{x} = Ax + Bu$$

$$x = \begin{pmatrix} u \\ r \end{pmatrix}$$

$$\text{entradas} = \begin{pmatrix} F_1 \\ F_2 \end{pmatrix}$$

$$\begin{pmatrix} F_1 \\ F_2 \end{pmatrix} = Gv \rightarrow G = B^{-1}$$

$$\begin{pmatrix} \dot{u} \\ \dot{r} \end{pmatrix} = \begin{pmatrix} \frac{X_u}{m - X_{\dot{u}}} & 0 \\ 0 & \frac{N_r}{I_{zz} - N_{\dot{r}}} \end{pmatrix} \begin{pmatrix} u \\ r \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$$

$$\dot{u} = \left(\frac{X_u}{m - X_{\dot{u}}}\right) u + v_1 \qquad (4)$$

$$\dot{r} = \left(\frac{N_r}{I_{zz} - N_{\dot{r}}}\right) r + v_2 \qquad (5)$$

7.3 SIMULAÇÕES DO MODELO HORIZONTAL

As equações desacopladas foram obtidas através do Matlab e são dadas a seguir:

$$\dot{x} = \begin{bmatrix} -0{,}0750 & 0 \\ 0 & -3{,}4417 \end{bmatrix} x + \begin{bmatrix} 0{,}2167 & 0{,}2167 \\ -2{,}2945 & 2{,}2945 \end{bmatrix} u$$

$$y = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x$$

Chamaremos de G_1 e G_2 as equações discretas para as entradas u_1 e u_2 respectivamente. Tendo posse das equações discretas, obtidas em Matlab [1], para cada saída podemos traçar os gráficos da resposta ao degrau que nosso sistema real deve apresentar em malha aberta.

Para nosso G_1 discretizado, a resposta ao degrau é apresentada pela figura 9.

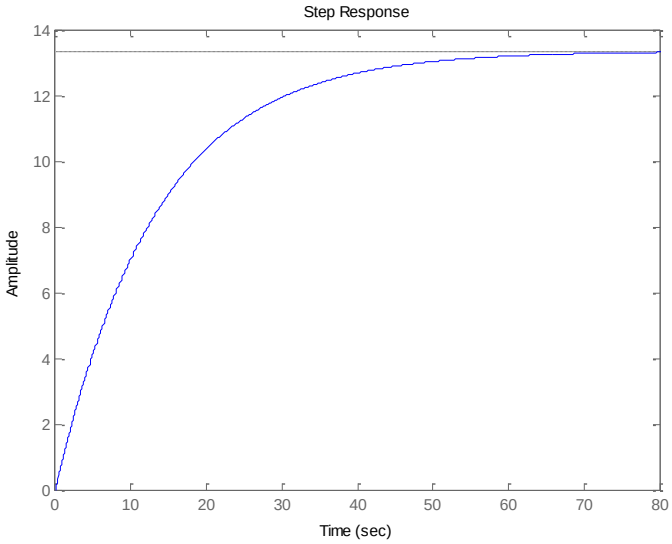

Figura 29 - Resposta ao degrau de G1 no plano z

Para o G_2 discretizado, a resposta ao degrau é apresentada pela figura 10.

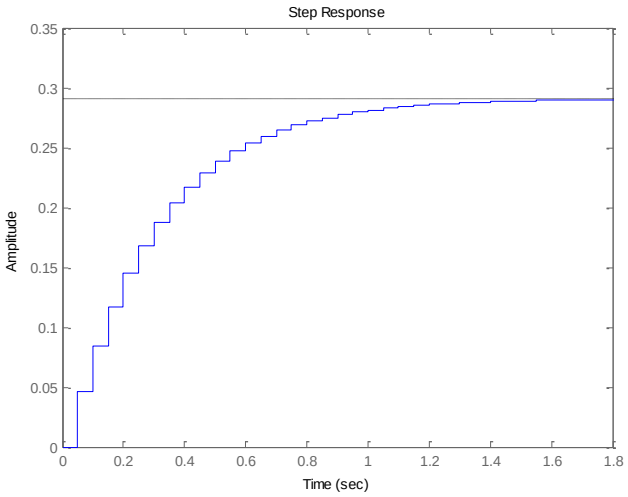

Figura 30 - Resposta ao degrau de G2 no plano z

A seguir fazemos análise mais aprofundada de cada função de transferência, G_1 e G_2, de acordo com a teoria de controle de modo que possamos compreender melhor esse comportamento mostrado na resposta ao degrau.

No plano s, temos que a resposta ao degrau de G_1 e G_2 é apresentada pela figura a seguir.

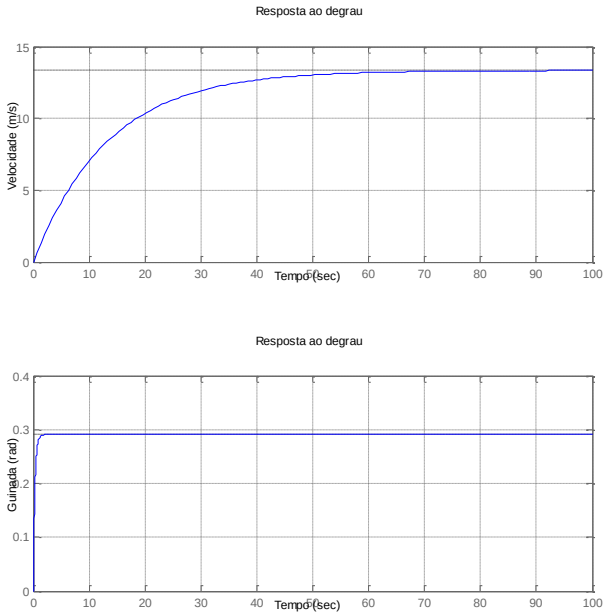

Figura 31 - Resposta ao degrau de G1 e G2 no plano s

Notamos que a resposta ao degrau da guinada é muito mais rápida do que a velocidade linear para atingir seu ponto de regime permanente.

Os lugares da raiz para G_1 e G_2 respectivamente é apresentada pela figura seguinte.

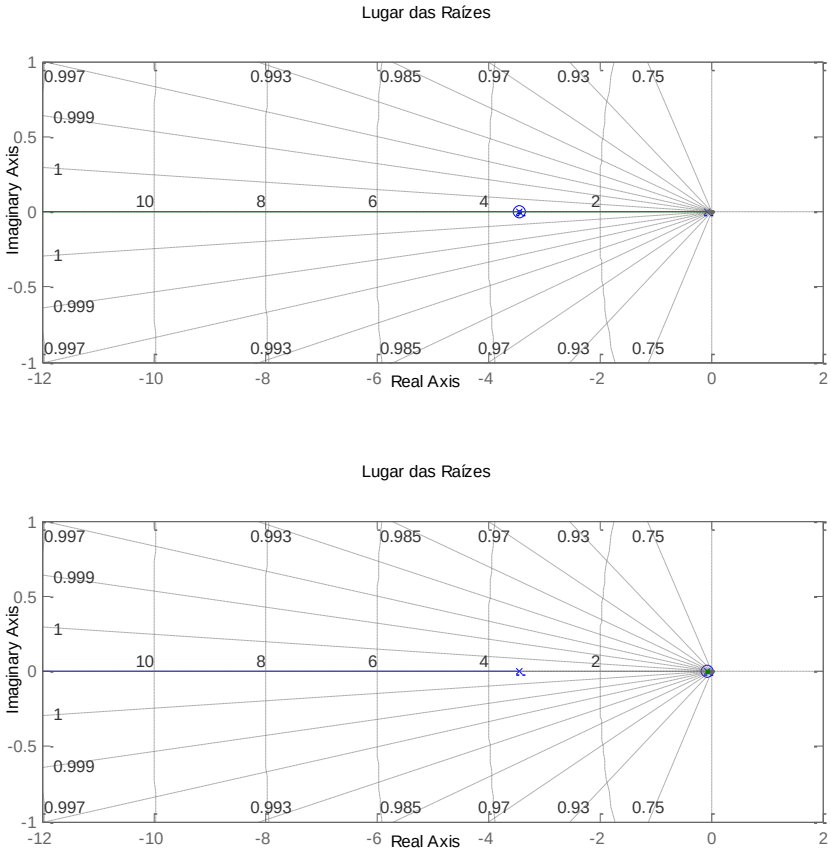

Figura 32 - Lugar das raízes de G1 e G2

Na qual notamos que para a função G_1 temos um zero em s = -3.45 e em G_2 temos um zero em s = -0.075.

Os pólos de G_1 estão em s1 = -3.45 e s2 = -0.075. Os pólos de G_2 estão em s_1 = -3.45 e s_2= -0.075, evidentemente os mesmos de G_1 pois ambas funções de transferência possuem o mesmo denominador.

A partir desses pólos e zeros e dos gráficos notamos que as duas funções de transferência, G_1 e G_2, são muito semelhantes. A diferença dessas duas plantas evidenciada pelo lugar das raízes está em como seus pólos e zeros se combinam, sendo que em G_1o zero cancela o pólo mais rápido e em G_2o zero cancela o pólo mais lento, tornando a resposta de G_2em torno de 100 vezes mais rápida que de G_1.

Prosseguindo com as análises temos que os Diagramas de Bode para G_1 e G_2 conforme segue.

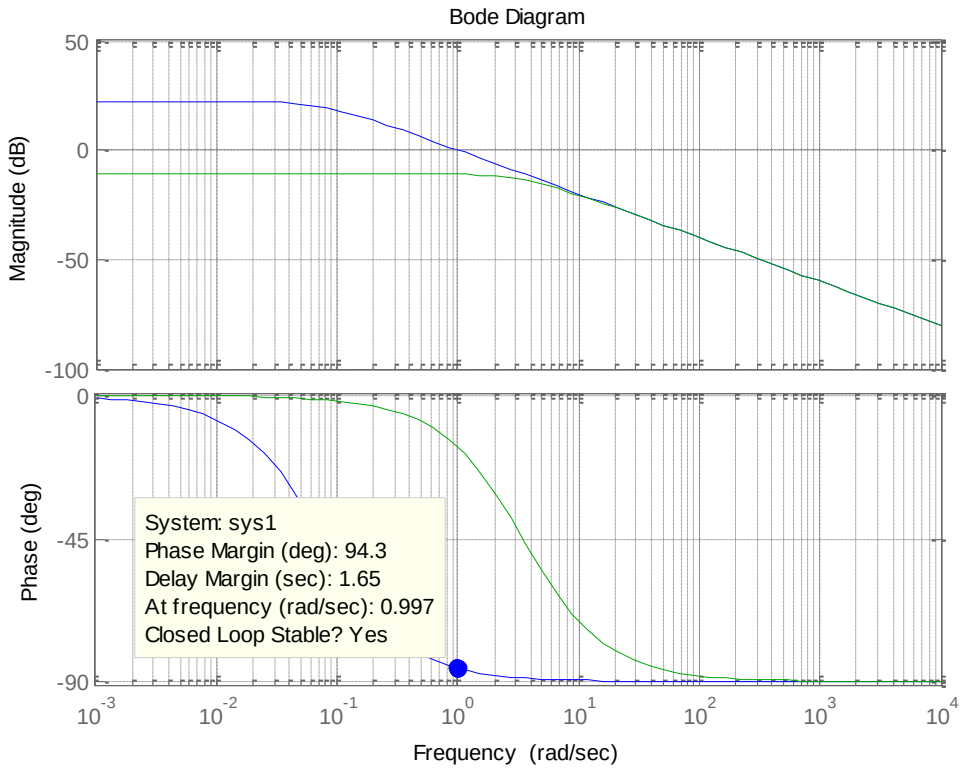

Figura 33 - Diagramas de bode de G1 e G2

Observando o Diagrama para a função de transferência G_1temos que a margem de ganho evidenciada pelo gráfico é infinito pelo sistema ser de primeira ordem, devido ao cancelamento de um pólo com um zero, e nunca atingir os 180 graus. A margem de fase que se observa para G_1é de 94.3 graus em aproximadamente 1 radiano.

Para G_2temos que observando o Diagrama de Bode tanto a margem de ganho quanto a margem de fase valem infinito, pois, o gráfico de fase nunca chega a 180 graus e o gráfico de ganho nunca cruza o zero.

Em seguida analisamos o Diagrama de Nyquist para G_1 e G_2, respectivamente pelas figuras 34 e 35.

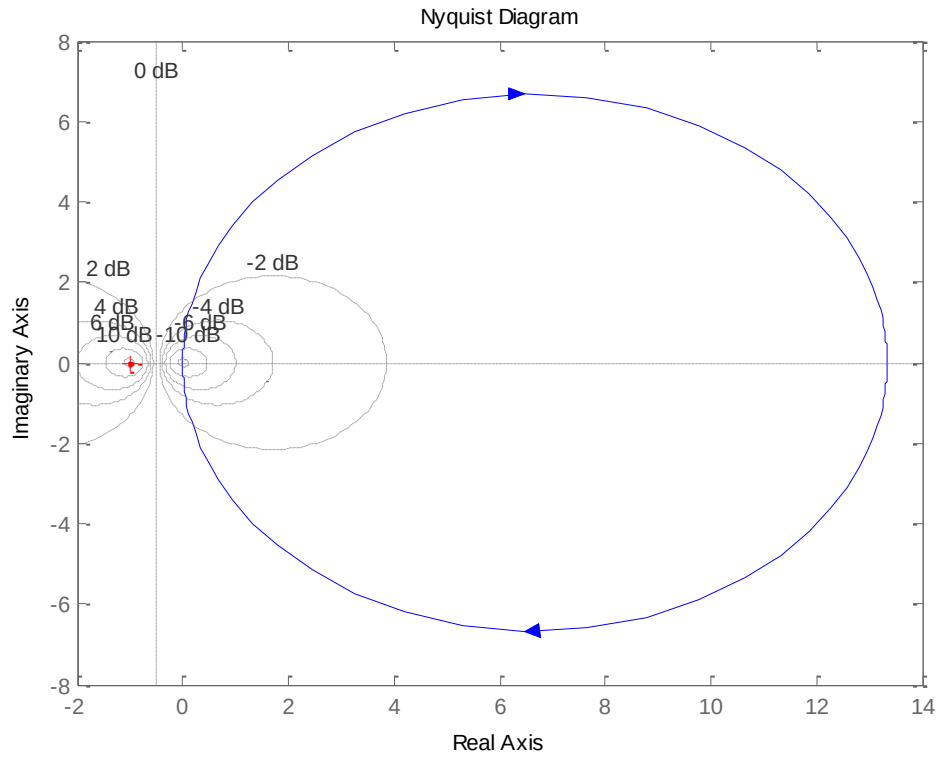

Figura 34 - Diagrama de Nyquist de G1

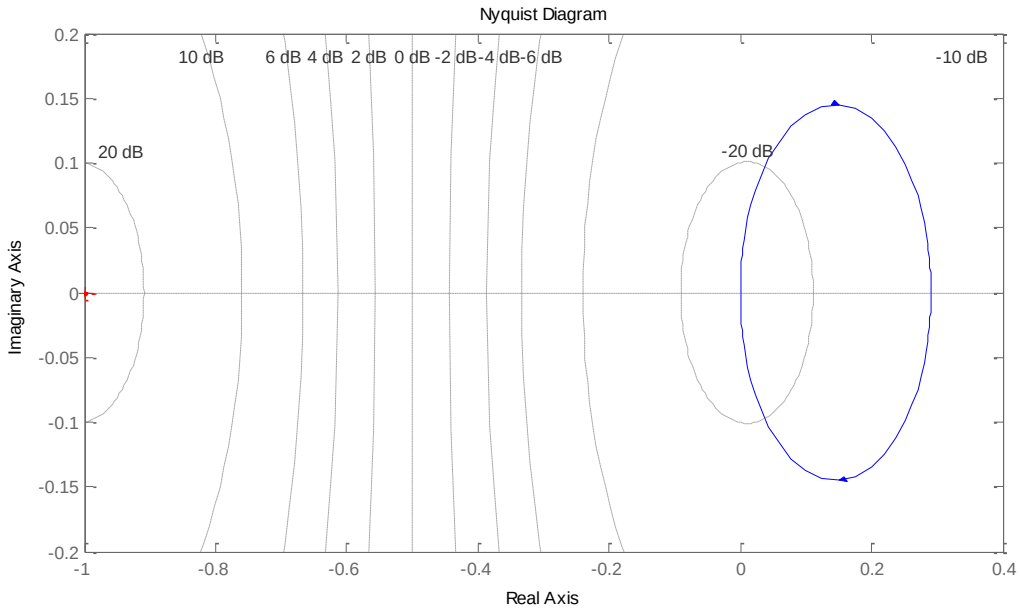

Figura 35 - Diagrama de Nyquist de G2

Analisando ambos os gráficos notamos que tanto G_1 quanto G_2 tem o grau do polinômio do denominador menos o do numerador igual a 1 devido à aproximação da origem pelo quarto quadrante. Confirmamos também que G_1 e G_2são do tipo zero.

Analisando o Critério de Nyquist, para a robustez de estabilidade, temos que tanto para G_1 quanto para G_2 o gráfico de Nyquist dá uma volta completa no sentido horário, porém para nenhuma das duas funções de transferência temos o envolvimento da origem. Portanto, tanto G_1 quanto G_2 atendem o Critério de Nyquist, para robustez de estabilidade, e podem ser ditos estáveis.

Prosseguindo com a análise, temos as Diagramas de Nichols para G_1 e G_2 na figura 36.

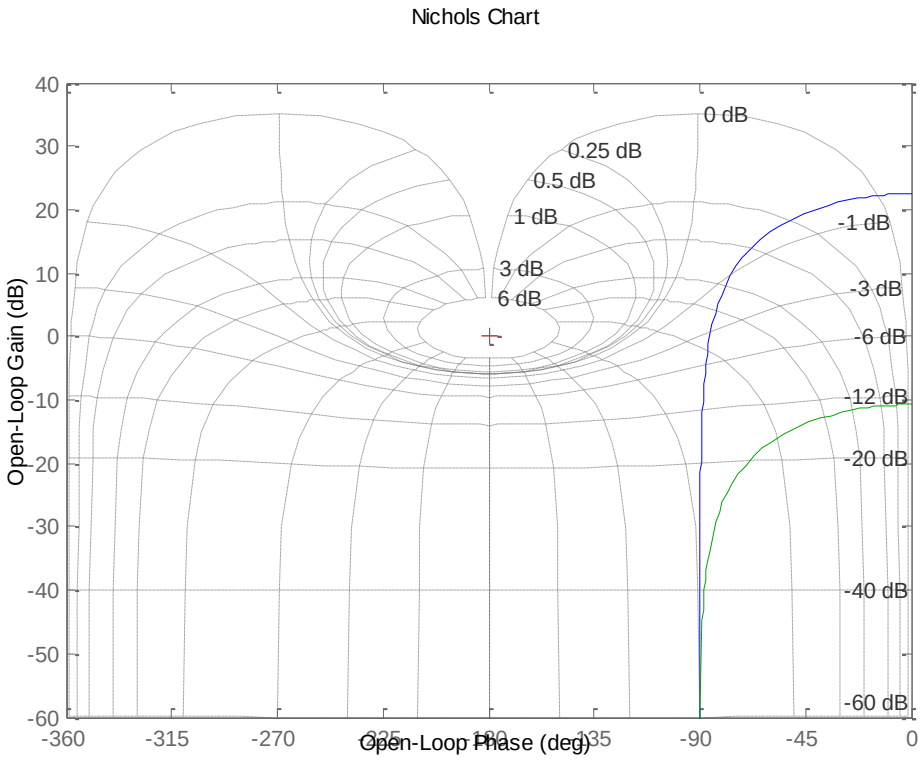

Figura 36 - Diagrama de Nichols de G1 e G2

A partir dos gráficos do Diagrama de Nichols observamos diretamente as margens de ganho e de fase, já expostas anteriormente com os gráficos de Bode.

8 RASTREAMENTO

8.1 PROBLEMA DE RASTREAMENTO

Até essa etapa do projeto, apenas o controle vertical havia sido realizado. O controle de profundidade foi baseado na teoria de controle robusto através das técnicas de QFT.

Agora passamos para uma etapa essencial que é o controle horizontal. Tal controle será responsável pelo avanço e guinada do ROV.

Uma vez que se trata de um sistema com duas entradas, optou-se por desenvolver o controle a partir da Teoria de Regulação e do Rastreamento. A teoria do rastreamento estuda o problema de projetar um sistema de controle que garanta que a saída do sistema siga um sinal de referência r(t) com um erro assintótico nulo. A teoria da regulação trata do problema de projetar um sistema de controle capaz de eliminar assintoticamente o efeito de uma perturbação na saída, uma vez que o problema de regulação é um caso particular do problema do problema de rastreamento [4], pretendemos resolver ambos.

Queremos resolver o problema de rastreamento para o sistema

$$\dot{x} = \begin{bmatrix} -0{,}0750 & 0 \\ 0 & -3{,}4417 \end{bmatrix} x + \begin{bmatrix} 0{,}2167 & 0{,}2167 \\ -2{,}2945 & 2{,}2945 \end{bmatrix} u$$

$$y = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x$$

Que pode ser reescrito da seguinte forma:

$$\dot{x_1} = -0{,}0750 + 0{,}2167u_1 + 0{,}2167u_2$$

$$\dot{x_2} = -3{,}4417x_2 - 2{,}2945u_1 + 2{,}2945u_2$$

$$y_1 = x_1$$

$$y_2 = x_2$$

Mostraremos que o problema de rastreamento é solúvel. Para isso utilizaremos o Teorema 9 de [4].

O sistema possui grau relativo:

- Para y_1

$$\dot{y_1} = \dot{x_1} = -0{,}0750 + 0{,}2167u_1 + 0{,}2167u_2$$

O grau relativo $\rho_1 = 1$, pois ao derivarmos y_1 apenas uma vez, já obtemos uma dependência explícita da entrada u. Além disso, temos que:

$$a_1 = -0{,}0750$$

$$A_1 = [0{,}2167 \quad 0{,}2167]$$

- Para y_2

$$\dot{y_2} = \dot{x_2} = -3{,}4417x_2 - 2{,}2945u_1 + 2{,}2945u_2$$

O grau relativo $\rho_2 = 1$, pois ao derivarmos y_2 apenas uma vez, já obtemos uma dependência explícita da entrada u. Além disso, temos que:

$$a_2 = -3{,}4417$$

$$A_2 = [-2{,}2945 \quad 2{,}2945]$$

Finalmente temos

$$a = \begin{bmatrix} -0{,}0750 \\ -3{,}4417 \end{bmatrix}$$

$$A = \begin{bmatrix} 0{,}2167 & 0{,}2167 \\ -2{,}2945 & 2{,}2945 \end{bmatrix}$$

E podemos calcular $\det(A) = 0{,}9944 \neq 0$, portando a matriz de desacoplamento A(x) é não singular.

Finalmente, $\sum \rho_i = \rho_1 + \rho_2 = 1 + 1 = 2$, que é igual à dimensão do espaço.

Portanto, de acordo com [4] o problema de rastreamento é solúvel.

8.2 SIMULAÇÃO DO RASTREAMENTO

Após mostrarmos que o problema de rastreamento é solúvel partimos para sua implementação e resolução em Matlab conforme programa no apêndice 13.1.

8.2.1 Simulação sem Distúrbio

Primeiro realizamos um teste com o sistema sem ruídos. O diagrama em simulink utilizado é o que segue.

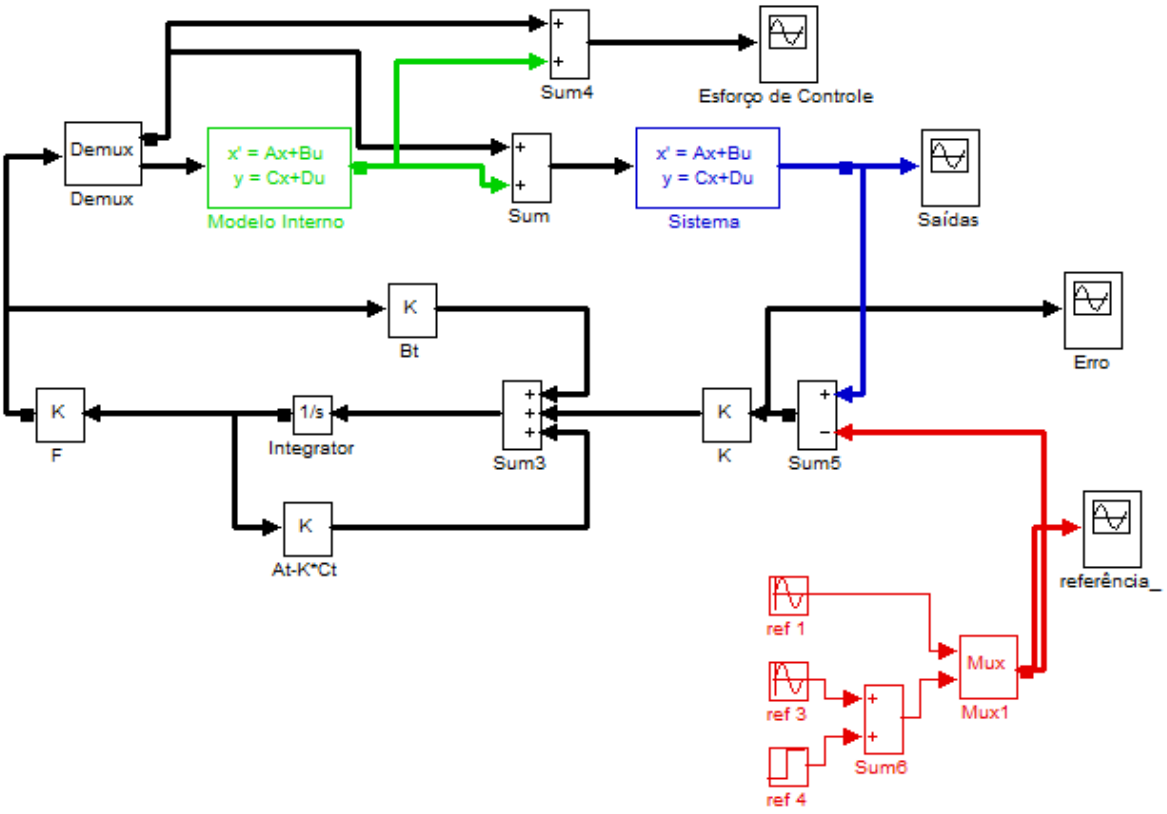

Figura 37 – Diagrama em Simulink para o Rastreamento

Colocamos uma onda senoidal como sinal de referência para cada entrada u a fim de avaliar o desempenho do controlador, mas as referências que usaremos na prática têm um comportamento menos oscilatório.

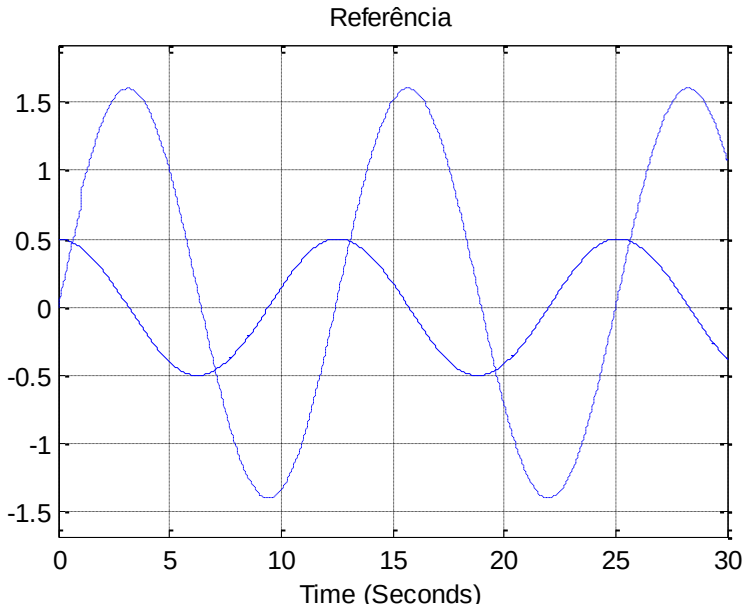

Figura 38 – Referência

O sinal de saída obtido foi extremante satisfatório, muito próximo da referência. No início notamos que o sinal do erro chega próximo de -0.5 mas, quando o sistema estabiliza o erro é ínfimo, menos que 0.1 em módulo, conforme figuras a seguir:

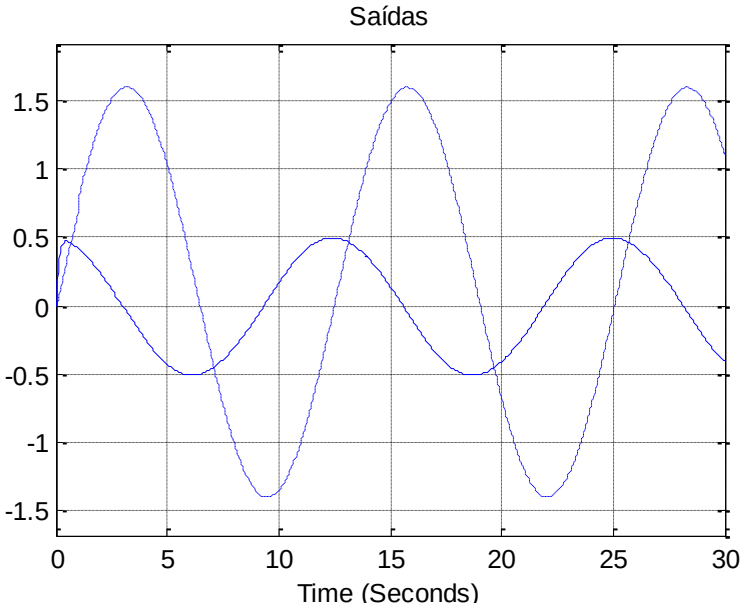

Figura 39 – Saídas u_1 e u_2

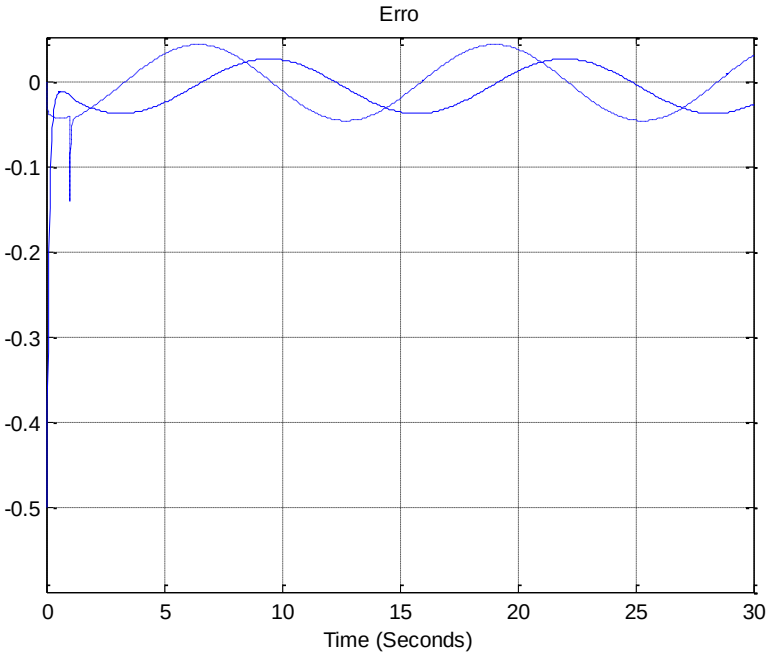

Figura 40 – Sinal do erro

Na figura a seguir podemos observar o esforço de controle. Inicialmente ele é maior e diminui conforme a estabilização do sistema.

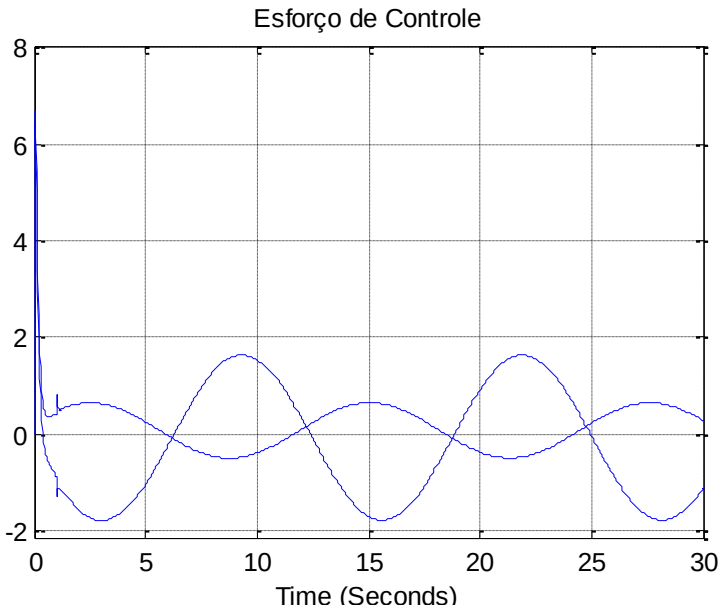

Figura 41 – Esforço de Controle

8.2.2 Simulação com Distúrbio

Acrescentou-se um sinal de distúrbio ao diagrama do simulink.

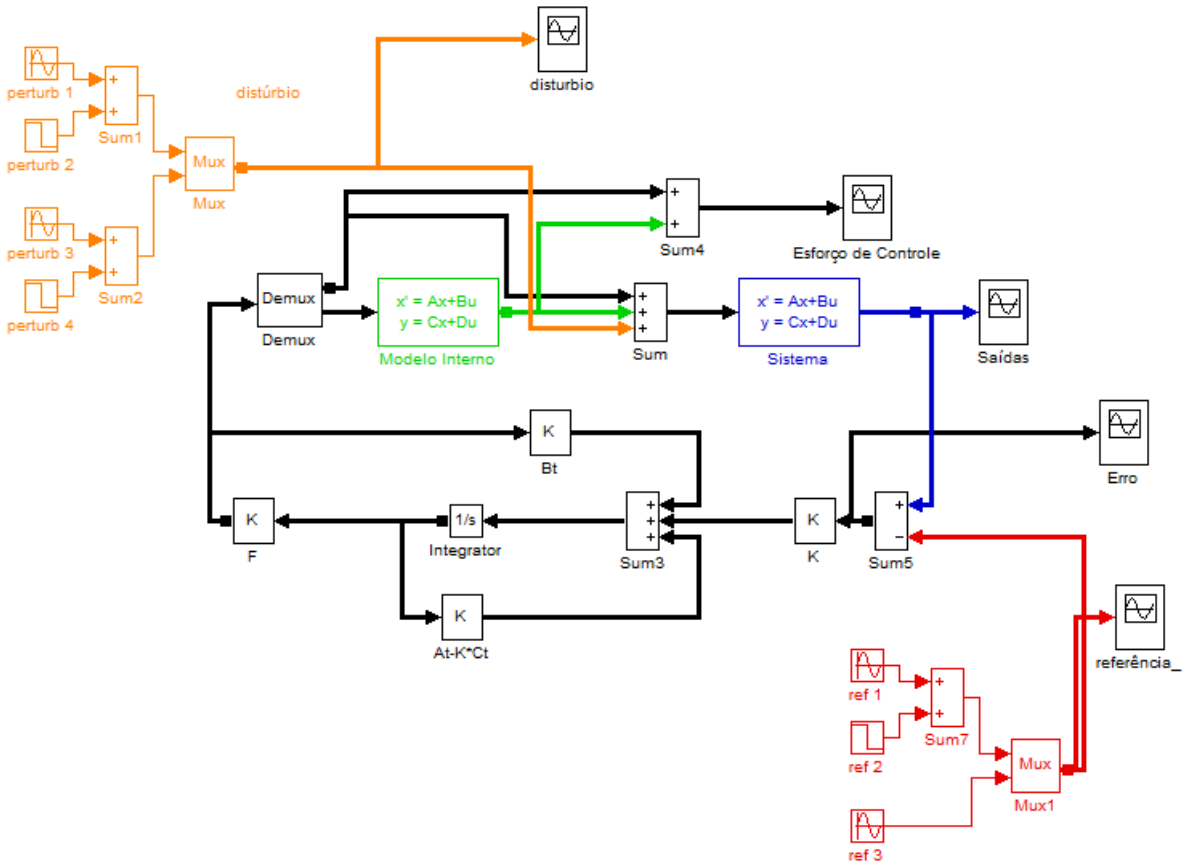

Figura 42 – Diagrama para simulação de rastreamento com distúrbio

Novamente colocamos uma onda senoidal como sinal de referência para cada entrada u entretanto, dessa vez acrescentou-se um sinal de distúrbio para cada uma das entradas do sistema. Note que os sinais de distúrbios possuem amplitudes muito maiores do que as dos sinais de referência.

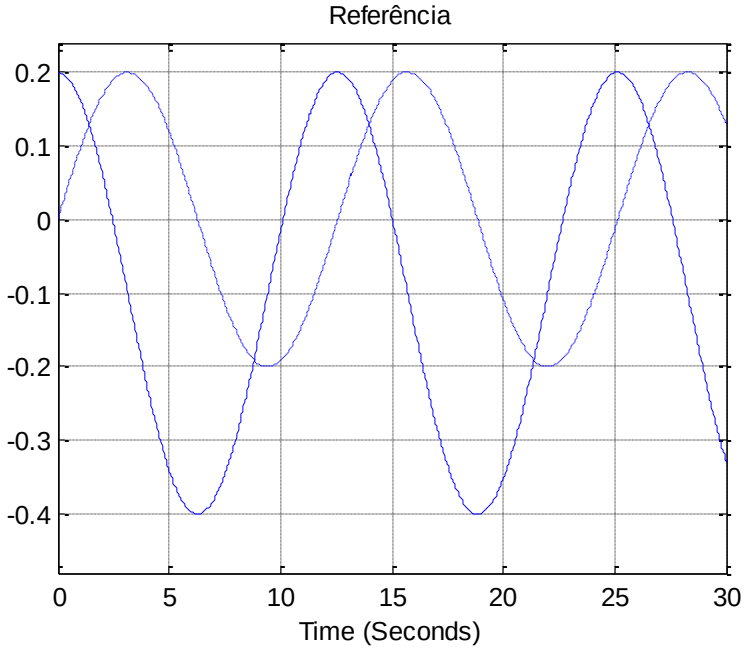

Figura 43 – Referência

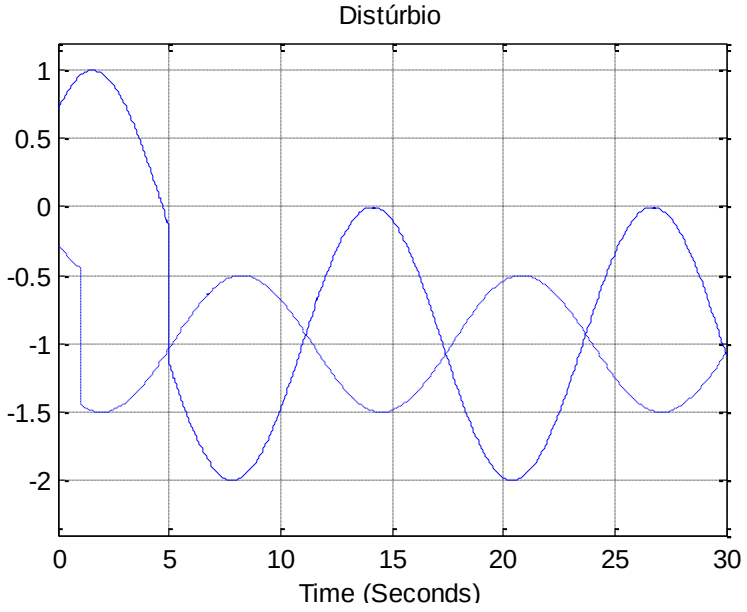

Figura 44 - Distúrbio

Os sinais de saída apresentaram amplitudes maiores do que os sinais de referência, mas ainda assim é possível notar que o rastreamento foi adequado mesmo com um ruído tão alto.

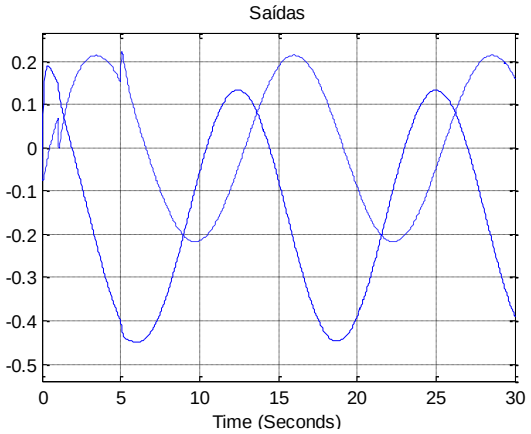

Figura 45 - Saídas

O erro alcançou inicialmente picos de -0.2 e 0.1, muito próximo do valor de referência. Entretanto a saída não apresentou uma discrepância tão alta. Provavelmente isso se deve aos sinais de distúrbio.

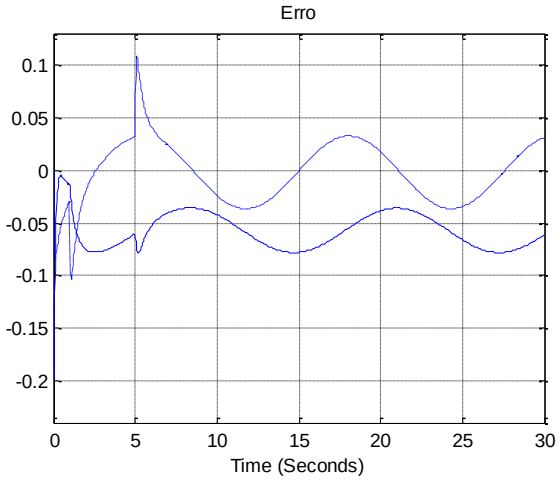

Figura 46 – Erro de rastreamento

O esforço de controle surpreendentemente não foi muito diferente do caso sem ruído, conforme pode ser visto a seguir:

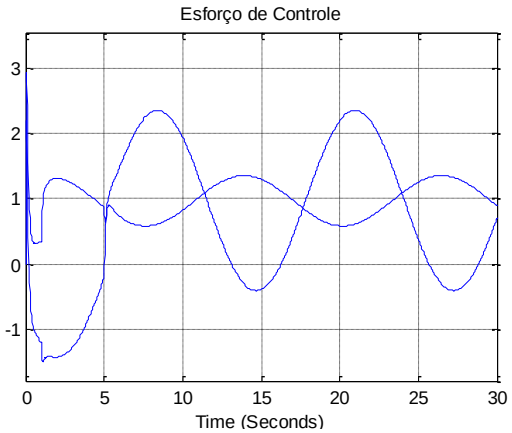

Figura 47 – Esforço de Controle

9 FINALIZAÇÃO DA MONTAGEM DO ROV

Após a troca de componentes, remodelagem e resolução do problema de rastreamento, passou-se à etapa de montagem do ROV. Até o presente momento o ROV não havia sido montado por completo, os testes que foram realizados em água não possuíam o cabo umbilical que serve para alimentar o ROV, conectá-lo à internet e permitir a utilização do Joystick.

Figura 48 – Última alteração física do ROV em 2015

9.1 HÉLICES

Um dos impasses encontrados para a montagem do ROV foi o encaixe dos hélices nos motores. Os motores e *spinners* (adaptadores de hélice) adquiridos são utilizados para aeromodelismo. As hélices utilizadas em aeromodelismo, diferentemente dos hélices para submarino, possuem um eixo curto, e portanto, são facilmente ajustáveis aos spinners, responsáveis por conectar o hélice ao motor. Entretanto, nossos hélices não encaixavam nos spinners, tanto pelo comprimento do eixo como pelo diâmetro do orifício em nosso hélice, como é possível notar nas imagens a seguir.

Figura 49 – Hélice de aeromodelismo à esquerda. Hélice para ROV à direita.

Figura 50 – Encaixe da hélice de aeromodelismo

Figura 51 – Hélice do ROV e Spinner

Inicialmente pensou-se em adquirir novos spinners que possuíssem um eixo mais cumprido. Foram visitadas lojas de aeromodelismo e tais spinners não existiam nas lojas, já que o modelo de hélice utilizado no aeromodelismo era diferente. Passou-se então à procura de uma alternativa para resolver o problema. A compra de novos hélices foi descartada uma vez que a geometria do ROV não permitiria um outro modelo de hélice, no caso, um hélice maior. Pensou-se então em estender o eixo do spinner com um parafuso, mas a alternativa se tornou inviável tanto pela descontinuidade da rosca que seria gerada quanto pela incerteza de que tal resolução permitiria que os hélices ficassem fixos.

Após muito se pensar, alternativa encontrada para esse problema foi de uma simplicidade admirável. A solução foi utilizar um macho de mesmo diâmetro do eixo do spinner para rosquear o "eixo" interno do hélice. Com essa alternativa não seria necessário utilizar a parte "superior" do spinner para prender o hélice ao motor, o próprio rosqueamento já realizaria essa tarefa. O resultado por ser visto nas imagens a seguir.

Figura 52 – Hélice com rosca

Figura 53 – Hélice conectada ao Spinner

Figura 54 – Hélice acoplada ao motor sem e com o recipiente do motor

9.2 CABO UMBILICAL

Cabos umbilicais submarinos são cabos que conectam o ambiente terrestre à equipamentos submersos. Cabos umbilicais são utilizados para transmitir elétrica (alimentação, dados do equipamento para a superfície, etc) e fluidos como água e petróleo em plataformas de petróleo. O cabo umbilical no ROV será responsável pela alimentação dos motores e da placa Raspberry Pi, e conexão com o computador (cabo Ethernet).

Para a montagem do ROV foi primeiramente necessário decidir qual tipo de cabo seria necessário para construir o cabo umbilical. Notou-se que seriam necessários 11 cabos ao todo: duas perfurações para cada um dos quatro motores (alimentação e saída da placa para o motor), dois orifícios para as lanternas e um para o cabo Ethernet. Para evitar que houvessem 11 perfurações e 11 prensa cabos na tampa, optou-se pela utilização de um cabo de 4 vias. Assim, seria possível utilizar um cabo para cada dois motores. Foram adquiridos 7 metros desse cabo. A medida foi baseada na medida do cabo de Ethernet que é de 3 metros. Então serão 3,5m (50 cm de folga) para cada par de motores.

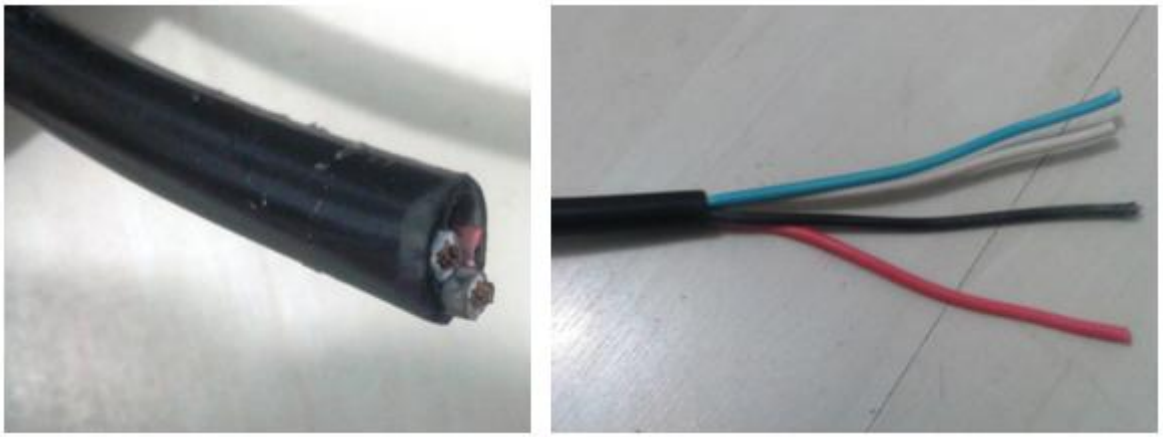

Figura 55 – Preparação do cabo

O cabo foi dividido em dois e então era necessário adaptá-lo para realizar a conexão com a ESC. Cada um dos fios internos de ambos os cabos foram descascados e dois a dois soldados ao conector mostrado na figura abaixo.

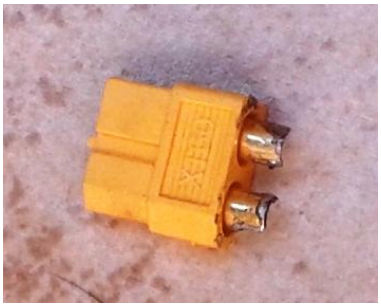

Figura 56 - Conector

Era necessário passar os fios pelos prensa cabos da tampa antes de realizar o trabalho de soldagem e acabamento, e uma vez soldados não seria possível remover os cabos. Esse trabalho finalizado é mostrado na figura a seguir.

Figura 57 – Cabos finalizados com conectores

9.3 CONEXÃO MOTORES E ESC

Durante a estruturação física do ROV, notou-se que os fios que conectavam ESC e motores eram extremamente curtos, o que impossibilitaria que a ESC ficasse do lado interno do ROV sem contato com a água como pode ser observado na foto abaixo.

Figura 58 – Problema do comprimento dos fios do motor e ESC

A solução imediata para esse problema era prolongar os fios dos motores ou da ESC. Como ambos são componentes caros e delicados, antes de qualquer procedimento foi feita uma pesquisa de qual seria a melhor maneira de prolongar esses fios. Durante as pesquisas e conversas em fóruns de aeromodelismo descobriu-se que muitos usuários tiveram problemas decorrentes da prolongação dos fios. Tais problemas advém da forma de funcionamento da ESC, através de pulsos PWM. Prolongar os fios faz com que a ESC não consiga fazer uma leitura exata dos pulsos, podendo acarretar em dificuldade para girar os motores, desde trancos até a parada de funcionamento do motor. Levando isso em consideração, foi necessário buscar outras alternativas para a resolução do problema.

A alternativa mais factível foi a realização de furos laterais no ROV, assim, não seria necessária a prolongação dos fios. Apesar de estranha à primeira vista, a resolução não traz desvantagens, o único porém é a quebra do padrão de se ter todos os furos na tampa e a dificuldade de manipular os cabos dos motores já dentro do ROV. As imagens a seguir mostra como ficou o ROV após os furos.

Figura 59 – Orifícios laterais

Passou-se então para a alocação dos motores e finalização da montagem do ROV. Nas imagens a seguir é possível ver o passo a passo dessa montagem. Além disso é possível observar que os conectores do motor ficam acessíveis dentro do ROV.

Figura 60 – Alocação dos motores

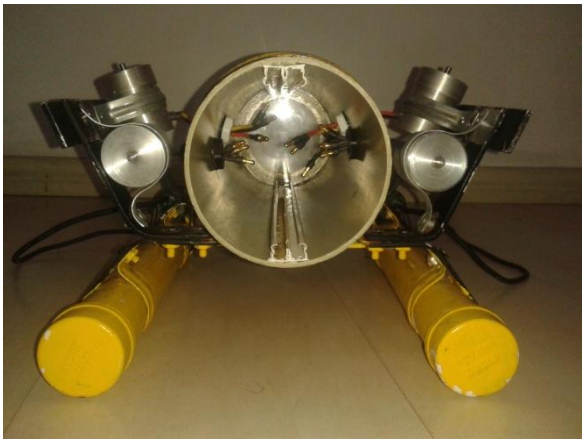

Figura 61 – Visão interna do ROV

Nessa etapa foram encaixados os hélices e o ROV adquiriu a forma a seguir.

Figura 62 – ROV com os Hélices

O circuito para funcionamento do motor é o seguinte: Conecta-se o motor à ESC, a ESC à fonte e também à Raspberry Pi. As imagens a seguir ilustram o passo a passo para a montagem do circuito. Nessa etapa, ainda não foi montado o circuito com a Raspberry pois ainda há a etapa de programação, e é necessário que a Raspberry esteja fora do ROV.

Figura 63 – Circuito Motor e Esc

Figura 64 – Finalização da montagem

Figura 65 – ROV Finalizado e com cabo umbilical

10 INTEGRAÇÃO

A etapa de integração pode ser considerada a etapa mais importante desse projeto. Até o momento, foram realizadas as programações da lanterna, joystick e do controle vertical. Apesar da aquisição da câmera, não existe uma programação voltada para o seu funcionamento no ROV. Nessa etapa pretende-se integrar essas programações e acrescentar a programação do controle horizontal, gerando um controle geral.

10.1 CONTROLE INTEGRADO PARA RASPBERRY PI

O controle integrado tem como objetivo unificar tudo o que já foi realizado até então. Até o presente momento foram programados diversos códigos de forma independente e sem a preocupação de uma integração. Os códigos realizados até agora tinham como interesse apenas o teste pontual dos dispositivos ou simulações, com exceção do código do Joystick que já tinha um objetivo prático.

Nessa etapa nos deparamos com os seguintes códigos estruturados de forma independente: o código do Joystick, os códigos de teste do sensor e dos motores (conforme anexo 14.1 e 14.2 respectivamente), o código do filtro de Kalman, que até o momento só havia sido utilizado para simulações em simulink, o código do controle vertical que ainda estava escrito para funcionar com o Arduino (placa de aquisição usada anteriormente). O código das lanternas deveria estar entre os códigos, entretanto o código não foi encontrado salvo entre os arquivos do projeto, então um novo código e teste deve ser elaborado.

10.2 CONTROLE VERTICAL PARA RASPBERRY PI

A programação do controle vertical sofre duas alterações, a primeira é referente ao fluxo de dados. Quando o controle vertical foi realizado, considerou-se que o fluxo de dados se daria entre os motores e o arduino. Agora que utilizamos as ESCs e a Raspberry Pi, é necessário corrigir a forma como essa troca de dados é realizada. A segunda alteração está relacionada aos novos parâmetros obtidos após a remodelagem.

Desprendeu-se um tempo considerável para o entendimento de como era realizado o fluxo de dados no Arduino para então programar a passagem desse fluxo de dados para a Raspberry Pi. O código obtido até o momento encontra-se no Apêndice 13.2. Note que a função de controle utilizada ainda é a função obtida antes da remodelagem. Após a remodelagem foram feitas as simulações de comportamento do sistema mas não foi realizado um novo controle vertical, apenas o horizontal. A não realização de um novo controle vertical se deu por não estar dentro do escopo e do cronograma dessa fase do projeto, além disso, na época em que foi feito, o controle vertical foi realizado com a utilização da técnica QFT e da ferramenta SISO-QFTIT. Tentou-se entender a utilização da ferramenta, mas a falta de conhecimento sobre controle robusto impossibilitou a realização do controle. Esse estudo demandaria muito tempo e atrasaria o cronograma dando a impressão de que um trabalho já realizado estava sendo refeito. Então para prosseguir o projeto do especificado no cronograma, optou-se por não realizar essa etapa novamente.

10.3 ESTRUTURAÇÃO DA INTEGRAÇÃO

Um dos impasses dessa etapa foi conceber como efetivar a união dos códigos e como estruturar um código teste para um código funcional, ou seja, um código que efetivamente executasse seu papel para o projeto do ROV.

O único código que até o momento estava ligado diretamente ao ROV era o código do Joystick. O Joystick que tínhamos até o presente momento era o que podemos ver na imagem a seguir.

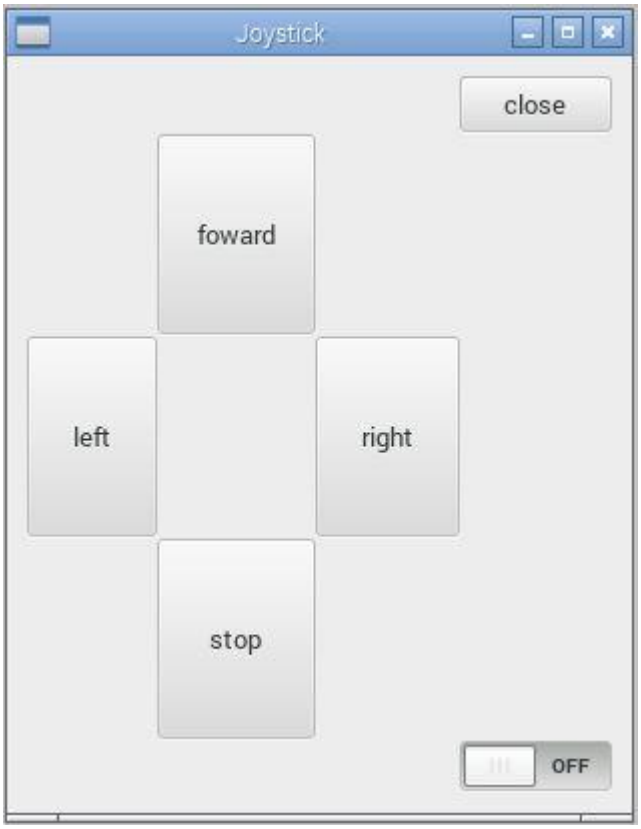

Figura 66 - Joystick

O Joystick apresentado era apenas visual e não funcional, já que quando foi construído ainda não havia sido realizado o teste dos motores. Além disso a forma como o Joystick foi pensado não refletia a realidade do que se quer realizar no projeto. Tanto para o controle horizontal quanto para o controle vertical quer-se construir um controle autônomo, ou seja, queremos informar para o ROV em que posição ele deve chegar e a partir daí ele se desloca para a posição. A forma como o Joystick foi pensada não contempla o controle autônomo, ele apenas informa onde quero que o motor esteja em um momento imediato.

Partiu-se desse Joystick para a integração do ROV, já que ele apresenta a interface gráfica do ROV com o usuário.

10.3.1 Integração Joystick e Controle

10.3.1.1 Controle Horizontal

Enquanto tentava-se realizar a integração do Joystick com o controle horizontal, que até o presente momento não havia sido escrito, chegou-se a um grande impasse: o controle horizontal foi pensado como uma resolução do problema de rastreamento, ou seja o sistema deveria receber uma função como entrada que seria seguida pelo controle horizontal. Entretanto não foi encontrada uma forma de o usuário fornecer uma função com ponto de partida e de chegada para ser seguida pelo ROV.

Uma das possibilidades para resolver esse impasse era deixar eplícitas algumas funções dentre as quais o usuário poderia escolher e o ROV seguiria. Porém ainda assim foi inviável prescrever de antemão uma função, pois precisaríamos de um gerador de funções ou trabalhar com o Matlab, por exemplo, para que pudéssemos fornecer uma função a ser seguida pelo sistema.

10.3.1.2 Controle Vertical

Passou-se então a pensar em como o controle vertical poderia ser implementado partindo do Joystick.A solução foi permitir que o usuário digitasse a profundidade desejada no Joystick e então, ao clicar em um botão no próprio Joystick, daria início ao controle vertical. O código do controle vertical pode ser visto no Apêndice 13.2. Note que nesse código o uso dos sensores já está integrado ao controle.

O Joystick foi então reelaborado para receber essa função e permitir a integração do controle vertical com o Joystick.

10.3.2 Integração Joystick e Lanterna

Conforme já foi citado, o código para as lanternas não foi encontrado, entretando um novo código foi simples de ser elaborado e foi acoplado ao Joystick com a função de botão Switch, que permite que liguemos e desliguemos as lanternas.

Na figura a seguir é possível visualizar o Joystick que supre as necessidades do controle vertical e da lanterna.

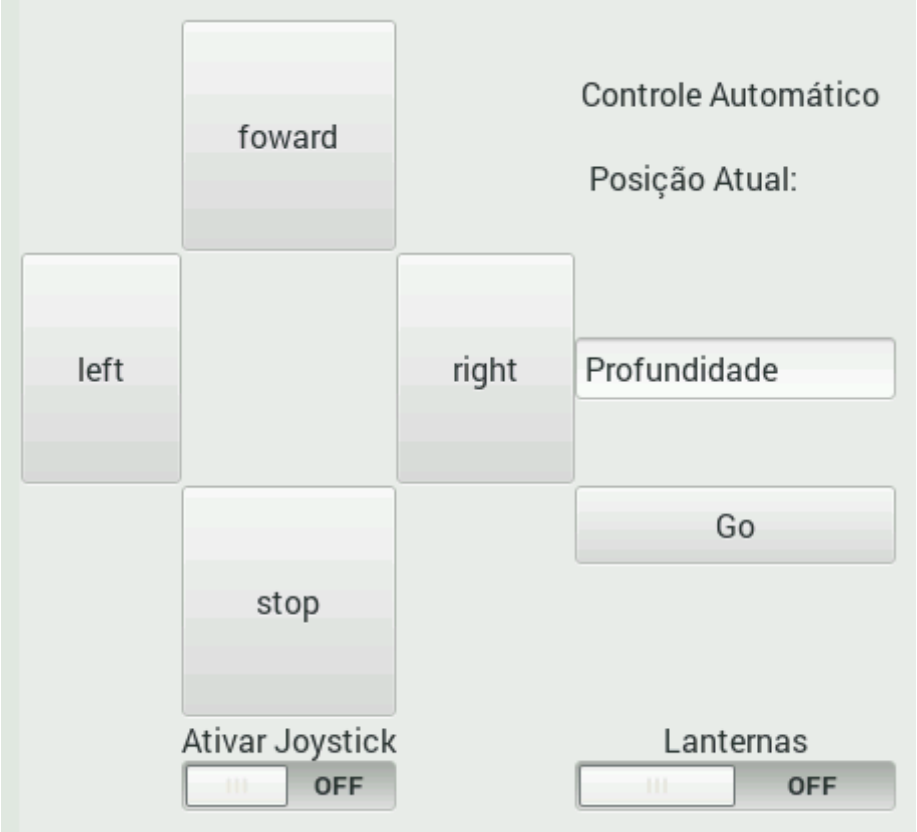

Figura 67 - Controle vertical e Joystick

10.3.3 Integração Joystick e Sensores

Os dados dos sensores são usados para a realização do controle vertical, na etapa em que se reescreveu o fluxo de dados do controle vertical, já houve a integração do sensor com o controle. Nesse momento, achou-se interessante e importante que o usuário saiba o que está ocorrendo com o ROV. Assim, adicionou-se ao Joystick a possibilidade do usuário visualizar os dados fornecidos pelos sensores.

O código de teste do Sensor GY-80 pode ser visto no Anexo 14.1. Um possível resultado para o sensor pode ser visto na imagem a seguir.

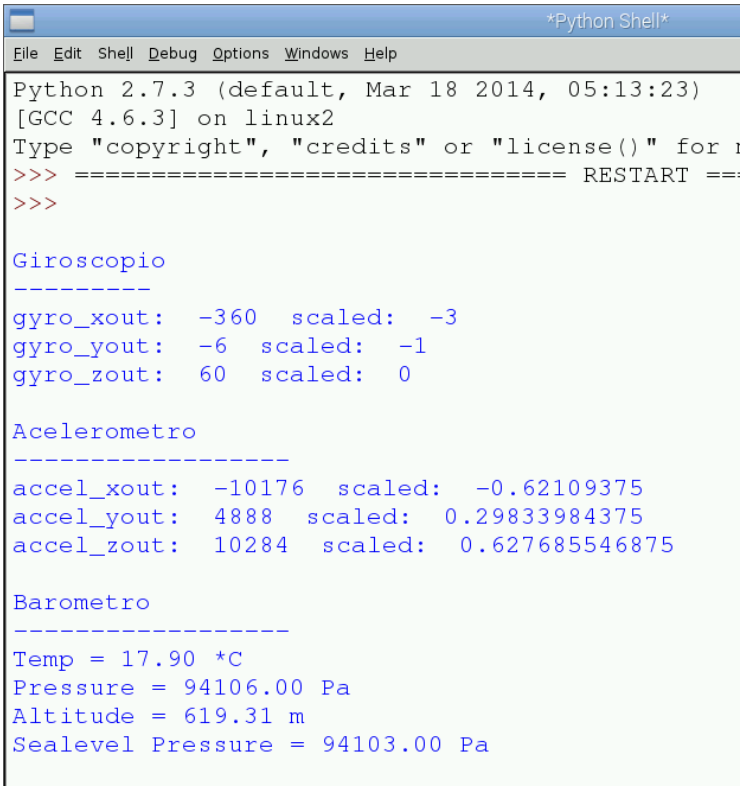

Figura 68 - Resultados do Sensor GY-80

A tela desenvolvida que mostra os dados do sensor para o usuário é a disposta na figura a seguir.

Figura 69 - Joystick finalizado

O código dessa integração, tela, leitura e processamento dos dados do sensor encontra-se no Apêndice 13.3.

Referente aos dados dos sensores, eles são atualizados quando clicamos no botão "Atualizar", nas próximas imagens podemos ver a resposta para a leitura do sensor e atualização de tela, os valores são similares, salvo pequenos desvios de medida.

```
Dados Atualizados
Giroscopio
---------
gyro_xout:  -398  scaled:  -4
gyro_yout:  -33  scaled:  -1
gyro_zout:  53  scaled:  0

Acelerometro
------------------
accel_xout:  -14000  scaled:  -0.8544921875
accel_yout:  7612  scaled:  0.464599609375
accel_zout:  1944  scaled:  0.11865234375

Barometro
------------------
Temp = 17.90 *C
Pressure = 94106.00 Pa
Altitude = 619.93 m
Sealevel Pressure = 94100.00 Pa
```

Figura 70 - Dados Atualizados

Figura 71 - Dados Atualizados e lanterna ligada

Na figura acima o comando de acender as lanternas está ativado. O comando de fato é funcional como podemos verificar na figura abaixo.

Figura 72 - Lanternas Ligadas

O Joystick atualizado é um joystick funcional, ou seja, não se trata mais de apenas uma tela estática. Os testes dos motores foram refeitos. Quando clicamos em "Frente", os dois motores horizontais recebem o comando de ligar, quando clicamos em "Esquerda" o motor da esquerda desliga e o motor da direita é ligado, fazendo com que o ROV curve-se para a esquerda, de maneira similar, quando clicamos em "Direita" o motor da direita desliga e o motor da esquerda é ligado, fazendo com que o ROV curve-se para a direita. Finalmente com o comando "Pare" todos os motores são desligados.

Essa etapa do trabalho desprendeu um esforço menos visual, já que se tratou de programação. Embora pareça simples quando observamos as imagens obtidas, o trabalho de se debugar os códigos e de utilizar o terminal do Linux, por exemplo, é demasiadamente árduo.

Até o presente momento não foi possível concluir de fato as etapas remanescentes, e novos problemas foram encontrados como a realização do controle horizontal, já que precisaríamos ter na entrada um sinal de referência, e não conseguimos isso com as programações até agora conhecidas. Teríamos que ter alguma interface com o Matlab para que talvez isso fosse possível. Um problema que ainda não havia sido avaliado é o da fixação do sensor. Até o momento ele ficaria solto dentro do ROV, entretando é necessário que o sensor fique fixo. Além disso, alguns fios, como o de alimentação da ESC ainda não foram passados pelo prensa cabos, já que precisaríamos cortá-los, além de obter fios maiores que os convencionais.

11 CONCLUSÃO

Durante a elaboração desse projeto muitos desafios foram superados. A quantidade de conhecimento adquirido a cada desafio foi diretamente proporcional às demandas exigidas. Foi extremamente proveitoso participar do projeto e conhecer desde as diversas facetas de um projeto como esse.

Pode-se elaborar desde a modelagem e parte física até o software. Todas as etapas trouxeram aprendizado, desde princípio as dificuldades Essa etapa do projeto exigiu um esforço que foi recompensado com demasiado aprendizado. Aprendeu-se a usar desde um software de análise hidrodinâmica como o Aqwa, até um componente mais próximo da engenharia elétrica como o Raspberry Pi. Além disso, os ganhos em se aprender a usar a linha de comando do Linux e a programar em uma linguagem de alto nível como o Python são inestimáveis.

Desenvolver o ROV trouxe conhecimentos das mais diversas áreas e permitiu a utilização de conhecimentos vistos em sala de aula de forma prática. O projeto mostrou-se completo em toda sua extensão, pois permitiu a vivência de um verdadeiro projeto de engenharia.

12 REFERÊNCIAS BIBLIOGRAFICAS

[1] MOREIRA, Diego Antonio; BARBOSA, Fernando Dos Santos. **Controle De Movimentação De Um Veículo Subaquático.** 2014. 136f. Trabalho de Conclusão de Curso - Escola Politécnica da Universidade de São Paulo, São Paulo, 2014.

[2] R. M. Gomes, J. B. Sousa et F. L. Pereira. **Modeling and Control of the IES Project ROV**. Porto.

[3] M. L. CENTENO, S. C. GOMES, I. M. PEREIRA, P. J. L. DREWS et S. BOTELHO. **Controle em Profundidade de um Veículo Subáquatico do Tipo ROV**. 2006.

[4] SILVA, P. S. P. **Controle Multivariável**. São Paulo, 2013. (Apostila)

[5] CASTRUCCI, P.B.L., BITAR, A. e SALES, R.M. **Controle Automático**. Rio de Janeiro, LTC, 2011.

[6] MARQUES, R. P. et al. **Apostila do Curso PTC2619 Laboratório de Automação.** São Paulo, 2016. (Apostila)

[7] S. A. M. R. A. R. E. M. A. M.S.M. ARAS, «DEVELOPMENT AND MODELING OF UNMANNED UNDERWATER REMOTELY OPERATED VEHICLE USING SYSTEM IDENTIFICATION FOR DEPTH CONTROLL,» 2013.

[8] W. Wang et C. Clark, «Autonomous Control foa a Differential Thrust ROV».

[9] M. L. CENTENO, S. C. GOMES, I. M. PEREIRA, P. J. L. DREWS et S. BOTELHO, «Controle em Profundidade de um Veículo Subáquatico do Tipo ROV,» 2006.

[10] H. F. P. N. R. M. SANTOS A. T., « PROJETO DE MINI SUBMARINO TELEGUIADO,» 2012.

[11] S. R. LEONARD N., «THE BELUGA PROJECT DEVELOPMENT OF A TESTED FOR AUTONOMOUS UNDERWATER VEHICLES,» 2010.

[12] F. T. I., «GUIDANCE AND CONTROL OF OCEAN VEHICLES,» 1994.

[13] A. M. TAVARES, «UM ESTUDO SOBRE A MODELAGEM E O CONTROLE DE VEÍCULOS SUBAQUÁTICOS NÃO TRIPULADOS,» 2003.

[14] http://www.e-voo.com (visitado em abril e maio de 2016)

[15] http://wiki.python.org.br/DocumentacaoPython#Artigos_e_Tutoriais (visitado em abril e maio de 2016)

[16] https://www.arduino.cc/en/Tutorial/HomePage (visitado em abril e maio de 2016)

[17] https://www.raspberrypi.org (visitado em 2016)

[18] https://learn.adafruit.com (visitado em 2016)

[19] https://docs.python.org/2/tutorial/ (visitado em 2016)

[20] http://python-gtk-3-tutorial.readthedocs.io/en/latest/ (visitado em junho de 2016)

[21] https://pythonhosted.org/RPIO/pwm_py.html (visitado em maio e junho de 2016)

[22] https://cdn-learn.adafruit.com/downloads/pdf/adafruits-raspberry-pi-lesson-3-network-setup.pdf (visitado em maio e junho de 2016)

[23] http://blog.filipeflop.com/embarcados/tutorial-raspberry-pi-linux.html (visitado em maio e junho de 2016)

[24] https://www.youtube.com/watch?v=TvINXo_2UrY (visitado em junho de 2016)

[25] https://www.youtube.com/watch?v=nuFf-LfJQmw (visitado em junho de 2016)

[26] http://astrobeano.blogspot.com.br/2014/01/gy-80-orientation-sensor-on-raspberry-pi.html (visitado em maio e junho de 2016)

[27] https://github.com/bitify/raspi (visitado em junho de 2016)

[28] http://myrobotlab.org/content/gy80 (visitado em junho de 2016)

[29] http://www.instructables.com/id/Raspberry-Pi-I2C-Python/?ALLSTEPS (visitado em junho de 2016)

[30] http://skpang.co.uk/blog/archives/575 (visitado em junho de 2016)

13 APÊNDICES

Este capítulo mostra todos os programas gerados pelos próprios formandos, responsáveis por este projeto.

13.1 CÓDIGO RASTREAMENTO

```
clear
close all
clc

%% Parâmetros
%Amortecimento Linear - D
% SIMULACAO NA AGUA
Xu = -0.346;
Yv = -0.547;
Zw = -0.845;
Kp = -0.3;
Mq = -0.3;
Nr = -0.3;
%Massa adicional - Ma
%MATRIZ INERCIA ADICIONAL
Xu_p = -1.091;
Yv_p = -3.891;
Zw_p = -4.048;
Kp_p = -0.021591;
Mq_p = -0.025051;
Nr_p = -0.025060;
%Amortecimento Quadrático - Dq
Xuu = 0;
Yvv = 0;
Zww = 0;
Kpp = 0;
Mqq = 0;
Nrr = 0;
%Inércia
Ixx = 0.046302;
Iyy = 0.035885;
Izz = 0.062105;
Ixy = 0;
Ixz = 0;
Iyz = 0;
%Massa
m = 3.524;
%Matrizes de Massa e Amortecimento
M = diag([m-Xu_p m-Yv_p m-Zw_p Ixx-Kp_p Iyy-Mq_p Izz-Nr_p]);
D = -diag([Xu Yv Zw Kp Mq Nr]);

%% Modelo Horizontal
%Distância dos motores em relação a y
D1 = 0.2;
D2 = -0.2;
%Matrizes de Estado
A = [Xu/(m-Xu_p) 0; 0 Nr/(Izz-Nr_p)];
```

```
B = [1/(m-Xu_p) 1/(m-Xu_p); -D1/(Izz-Nr_p) -D2/(Izz-Nr_p)];
C = [1 0; 0 1];
D = zeros(2);

zero = zeros(2);

%% Resolução do Problema de Rastreamento
Ao = [0 1; 0.0001 0];
Bo = [0; 1];
Co = [1 0];
Do = 0;

At = [A B*Co' [0;0]; zero Ao];
Bt = [B [0;0]; zero Bo];
Ct = [C zero];

Contr = CTRB(At,Bt);
rank(Contr);
Observ = OBSV(At, Ct);
rank(Observ);

[K,~,~] = lqr(At,Bt, 1e3*eye(4), eye(3));
[H,~,~] = lqr(At',Ct',1e3*eye(4),eye(2));

F = -K;
K = H';
```

13.2 CÓDIGO CONTROLE VERTICAL PARA RASPBERRY PI

```
#Código de controle de profundidade, projeto ROV.

import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BOARD)

from Adafruit_BMP085 import BMP085

bmp = BMP085(0x77)

entrada = 1
erro_1 = 0
erro_2 = 0
controle_1 = 0
controle_2 = 0
patm = 92853
pressmed = 0
profundidade = 0
controle = 0
i = 0

tempmed = bmp.readTemperature()
pressmed = bmp.readPressure()
altitude = bmp.readAltitude()
```

```
print"Temperature: %.2f C"% tempmed
print"Pressure: %.2f Pa"% pressmed
print"Altitude: %.2f m"% altitude

profundidade = (pressmed - patm)/1000
#p = patm + dgh
erro = entrada - profundidade
controle = 1.541*controle_1 - 0.59*controle_2 + 541*erro - 1000*erro_1 + 459.6*erro_2

if controle > 0
      i = 106*controle + 43
      if i > 255
            i = 255
      end

                  i= i/255
                  iniciaPWM(12, i) #motor1

                  #GPIO.setup(12, GPIO.OUT)
                  #MotorDir = GPIO.PWM(12, 0.5)
                  #MotorDir.start(1)

                  iniciaPWM(13, i) #motor2
                  #GPIO.setup(13, GPIO.OUT)
                  #MotorEsq = GPIO.PWM(13, 0.5)
                  #MotorEsq.start(1)

#Motor nao gira para ambos os lados,
#então os motores parar para o restante dos casos

else
      i=0
                  finalizaPWM(12)
                  finalizaPWM(13)
                  #MotorDir.stop()
                  #MotorEsq.stop()


end

controle_2 = controle_1
controle_1 = controle
erro_2 = erro_1
erro_1 = erro
```

13.3 CÓDIGO DE INTEGRAÇÃO SENSOR E JOYSTICK

```
# -*- coding: utf-8 -*-
import smbus
import math
import Adafruit_BMP.BMP085 as BMP085

import gi
gi.require_version('Gtk', '3.0')
from gi.repository import Gtk

# Registradores de gerenciamento de energia do sensor
power_mgmt_1 = 0x6b
power_mgmt_2 = 0x6c

#Funções do Sensor
def read_byte(adr):
   return bus.read_byte_data(address, adr)

def read_word(adr):
   high = bus.read_byte_data(address, adr)
   low = bus.read_byte_data(address, adr+1)
   val = (high << 8) + low
   return val

def read_word_2c(adr):
   val = read_word(adr)
   if (val >= 0x8000):
      return -((65535 - val) + 1)
   else:
      return val

def dist(a,b):
   return math.sqrt((a*a)+(b*b))

def get_y_rotation(x,y,z):
   radians = math.atan2(x, dist(y,z))
   return -math.degrees(radians)

def get_x_rotation(x,y,z):
   radians = math.atan2(y, dist(x,z))
   return math.degrees(radians)

# or bus = smbus.SMBus(0) para placas Revision 1
bus = smbus.SMBus(1)
address = 0x68  # Endereco lido para os Acelerometro e Giroscopio pelo i2cdetect

class joystick(Gtk.Window):

   def __init__(self):
      a = 1
      atual = 1
```

```
Gtk.Window.__init__(self, title="Joystick")
self.set_border_width(10)

grid = Gtk.Grid()
self.add(grid)

button1 = Gtk.Button("\n\n     Esquerda     \n\n")
button1.connect("clicked", self.on_click_me_clicked_lf, atual,a)
grid.add(button1)

button2 = Gtk.Button("\n\n    Frente    \n\n")
button2.connect("clicked", self.on_click_me_clicked_fw)
grid.attach(button2,1,-10,10,10)

button3 = Gtk.Button("\n\n    Pare!    \n\n")
button3.connect("clicked", self.on_click_me_clicked_stop)
grid.attach(button3,1,10,10,10)

button4 = Gtk.Button("\n\n     Direita     \n\n")
button4.connect("clicked", self.on_click_me_clicked_rg)
grid.attach(button4,15,0,10,5)

label0 = Gtk.Label('Ativar Movimentação \n por Joystick')
grid.attach(label0,1,200,1,1)

switch = Gtk.Switch()
switch.connect("notify::active", self.on_switch_activated)
grid.attach(switch,1,300,1,1)

#button5 = Gtk.Button("close")
#button5.connect("clicked", self.on_close_clicked)
#grid.attach(button5,100,-100,1,1)

label2 = Gtk.Label('Lanternas')
grid.attach(label2,600,200,1,1)

switch2 = Gtk.Switch()
switch2.connect("notify::active", self.on_switch2_activated)
grid.attach(switch2,600,300,1,1)

#label1 = Gtk.Label('actual angle: ' + str(a))
#grid.attach(label1,1000,-50,1,1)

#label2 = Gtk.Label('new angle: ' + str(a))
#grid.attach(label2,1000,-15,1,1)

label5 = Gtk.Label('CONTROLE AUTÔNOMO \n\n  DE PROFUNIDADE ')
grid.attach(label5,600,-2,1,1)

label6 = Gtk.Label('          ')
grid.attach(label6,500,-2,1,1)
```

```
        label7 = Gtk.Label('         ')
        grid.attach(label7,1000,-2,1,1)

        label8 = Gtk.Label('\n Profundidade Desejada:')
        grid.attach(label8,600,-1,1,1)

        entry1 = Gtk.Entry()
        entry1.set_text(" ")
        grid.attach(entry1,600,0,1,1)

        button6 = Gtk.Button(" Ir! ")
        button6.connect("clicked", self.on_click_me_clicked_go)
        grid.attach(button6,600,10,1,1)

        button7 = Gtk.Button(" Atualizar ")
        button7.connect("clicked", self.on_click_me_clicked_ad)
        grid.attach(button7,1100,300,1,1)

        #entry2 = Gtk.Entry()
        #entry2.set_text("Horizontal")
        #grid.attach(entry2,500,10,1,1)

        self.gyro_xout = read_word_2c(0x43)
        self.gyro_yout = read_word_2c(0x45)
        self.gyro_zout = read_word_2c(0x47)

        self.accel_xout = read_word_2c(0x3b)
        self.accel_yout = read_word_2c(0x3d)
        self.accel_zout = read_word_2c(0x3f)

        self.accel_xout_scaled = self.accel_xout / 16384.0
        self.accel_yout_scaled = self.accel_yout / 16384.0
        self.accel_zout_scaled = self.accel_zout / 16384.0

        sensor = BMP085.BMP085()

        label1 = Gtk.Label('\n DADOS DO SENSOR \n\n Temperatura (*C): '  +
str(format(sensor.read_temperature())) + ' \n Pressão (Pa): '   + str(format(sensor.read_pressure())) +
' \n Profundidade (m): '  + str(format(sensor.read_altitude())))
        grid.attach(label1,1100,-2,1,1)

        label3 = Gtk.Label('Giroscópio \n\n X: '  + str((self.gyro_xout / 131)) + ' \n Y: ' +
str((self.gyro_yout / 131)) + ' \n Z: ' + str((self.gyro_zout / 131)))
        grid.attach(label3,1100,0,1,1)

        label4 = Gtk.Label('\n Acelerômetro \n\n X: ' + str(self.accel_xout_scaled) + ' \n Y: '  +
str(self.accel_yout_scaled) + ' \n Z: '  + str(self.accel_zout_scaled))
        grid.attach(label4,1100,10,1,1)

    def on_click_me_clicked_lf(self, button,atual, a):
        global viraEsquerda
```

```
    viraEsquerda = 1
    print(atual + a)
    a = atual + a
    return atual + a

def on_click_me_clicked_fw(self, button):
    print("foward")

def on_click_me_clicked_rg(self, button):
    global viraDireita
    viraDireita = 1
    print("right")
    a = a + 1


def on_click_me_clicked_stop(self, button):
    print("stop")
    ##manda0(11)
    manda0(12)
    manda0(13)
    manda0(15)
    manda0(16)

def on_switch_activated(self, switch, gparam):
    if switch.get_active():
        state = "on"
        manda1(15)
        manda1(16)
    else:
        state = "off"
        manda0(15)
    manda0(16)
    print("Switch was turned", state)

def on_switch2_activated(self, switch, gparam):
    if switch2.get_active():
        state = "on"
        manda1(15)
        manda1(16)
        print("Lanternas Ligadas", state)
    else:
        state = "off"
        manda0(15)
        manda0(16)
        print("Lanternas Desligadas", state)

def on_click_me_clicked_go(self, button):
    print("Go: Controle automático vertical ativado")
    #FAZER CONTROLE VERTICAL

def on_click_me_clicked_ad(self, button):
    print("Dados Atualizados")
```

```
#DADOS DO SENSOR

self.gyro_xout = read_word_2c(0x43)
self.gyro_yout = read_word_2c(0x45)
self.gyro_zout = read_word_2c(0x47)

self.accel_xout = read_word_2c(0x3b)
self.accel_yout = read_word_2c(0x3d)
self.accel_zout = read_word_2c(0x3f)

self.accel_xout_scaled = self.accel_xout / 16384.0
self.accel_yout_scaled = self.accel_yout / 16384.0
self.accel_zout_scaled = self.accel_zout / 16384.0

sensor = BMP085.BMP085()

bus.write_byte_data(address, power_mgmt_1, 0)
print "Giroscopio"
print "----------"

gyro_xout = read_word_2c(0x43)
gyro_yout = read_word_2c(0x45)
gyro_zout = read_word_2c(0x47)

print "gyro_xout: ", gyro_xout, " scaled: ", (gyro_xout / 131)
print "gyro_yout: ", gyro_yout, " scaled: ", (gyro_yout / 131)
print "gyro_zout: ", gyro_zout, " scaled: ", (gyro_zout / 131)

print
print "Acelerometro"
print "------------------"

accel_xout = read_word_2c(0x3b)
accel_yout = read_word_2c(0x3d)
accel_zout = read_word_2c(0x3f)

accel_xout_scaled = accel_xout / 16384.0
accel_yout_scaled = accel_yout / 16384.0
accel_zout_scaled = accel_zout / 16384.0

print "accel_xout: ", accel_xout, " scaled: ", accel_xout_scaled
print "accel_yout: ", accel_yout, " scaled: ", accel_yout_scaled
print "accel_zout: ", accel_zout, " scaled: ", accel_zout_scaled

print
print "Barometro"
print "------------------"
sensor = BMP085.BMP085()
print 'Temp = {0:0.2f} *C'.format(sensor.read_temperature())
print 'Pressure = {0:0.2f} Pa'.format(sensor.read_pressure())
print 'Altitude = {0:0.2f} m'.format(sensor.read_altitude())
print 'Sealevel Pressure = {0:0.2f} Pa'.format(sensor.read_sealevel_pressure())
```

```
win = joystick()
win.connect("delete-event", Gtk.main_quit)
win.show_all()
Gtk.main()
```

13.4 CÓDIGO FILTRO DE KALMAN E CONTROLE

```
# -*- coding: utf-8 -*-
import smbus
import math
import Adafruit_BMP.BMP085 as BMP085
# OBS.: Utilizando essa biblioteca estamos admitindo que a frequencia da PWM é
#      de 50 Hz
# OBS.: Convencionamos que o Joystick representa a referência do nosso
#      sistema de controle.
# Está controlando apenas a parte horizontal do sistema
import RPIO
from RPIO import PWM
import numpy as np
from numpy.lialg import inv

from gi.repository import Gtk
import RPi.GPIO as GPIO
from motor2 import motor2
from joystick import joystick

GPIO.setmode(GPIO.BOARD)

# Registradores de gerenciamento de energia

power_mgmt_1 = 0x6b
power_mgmt_2 = 0x6c

def read_byte(adr):

   return bus.read_byte_data(address, adr)

def read_word(adr):

   high = bus.read_byte_data(address, adr)

   low = bus.read_byte_data(address, adr+1)
   val = (high << 8) + low
   return val
def read_word_2c(adr):
   val = read_word(adr)
   if (val >= 0x8000):
      return -((65535 - val) + 1)
   else:
      return val
```

```
def dist(a,b):
   return math.sqrt((a*a)+(b*b))

def get_y_rotation(x,y,z):
   radians = math.atan2(x, dist(y,z))
   return -math.degrees(radians)

def get_x_rotation(x,y,z):
   radians = math.atan2(y, dist(x,z))
   return math.degrees(radians)

######### JOYSTICK
#Rotina para jogar 1 para saida
def manda1(pino):
   GPIO.output(pino, GPIO.HIGH)
   return

#Rotina para jogar 0 para saida
def manda0(pino):
   GPIO.output(pino, GPIO.LOW)
   return

def guinada(atual, index):
   futuro = atual + index
   return futuro

def controleAvanco(chanel1, chanel2, avancoJoystick, avancoSensorKalman):

          global vx
          erro_1 = 0
          erro_2 = 0
          controle_1 = 0
          controle_2 = 0
          controle_3  = 0
          controle = 0
          i = 0

          #erro = avancoJoystick - y #y é a saida
          erro = avancoJoystick - avancoSensorKalman

          controle = (-0.001249*erro_2 – 0.000169*erro_1 + 0.001077*erro + controle_3 –
2.861*controle_2 + 2.722*controle_1) / 0.8611

          if controle > 0:
        i = 106*controle + 43
     if i > 255:
        i = 255
          else:
        i=0

     i = i/255
     i = i*100
```

```
      motor2.setW(i, i)

          if erro < 2:
        mymotor.stop()

############## Fim controleAvanco

############ Controle de guinada

#def controleGuinada(chanel, guinadaJoystick):
def controleGuinada(chanel1, chanel2, guinadaJoystick, guinadaSensorKalman):

          global vy
          erro_1 = 0
          controle_1 = 0
          controle_2 = 0
          patm = 92853
          controle = 0
          i = 0

          #erro = guinadaJoystick - y #y é a saida
          erro = guinadaJoystick - guinadaSensorKalman

          controle = (– 0.04651*erro_1 + 0.04637*erro + controle_2 – 1.861*controle_1) / 0.8611

          if controle > 0:
                    i = 106*controle + 43
                    if i > 255
                              i = 255
                    end
                    i = i/255
                    i = i*100
                    #iniciaPWM(chanel, i)
                    if viraDireita:
              motor2.setW(i, 0)
           if viraEsquerda:
              motor2.setW(0, i)
          else:
                    i=0
                    if viraDireita:
              motor2.setW(i, 0)
           if viraEsquerda:
              motor2.setW(0, i)
          end

      #Analisar qual valor de 'erro' é plausível
          if erro < 2:
        mymotor.stop()

#Fim controleGuinada#################################################################

#####KALMAN
```

```
# Função Filtro de Kalman

def filtroKalman(avanco, entrada):

        ##Informações sobre o modelo do sistema
        #Amortecimento Linear - D
        Xu = -0.346;
        Yv = -0.547;
        Zw = -0.845;
        Kp = -0.3;
        Mq = -0.3;
        Nr = -0.3;
        #Massa adicional - Ma
        Xu_p = -1.379;
        Yv_p = -3.791;
        Zw_p = -4.6;
        Kp_p = -0.022759;
        Mq_p = -0.025677;
        Nr_p = -0.026896;
        #Amortecimento Quadrático - Dq
        Xuu = 0;
        Yvv = 0;
        Zww = 0;
        Kpp = 0;
        Mqq = 0;
        Nrr = 0;
        #Inércia
        Ixx = 0.059301;
        Iyy = 0.040323;
        Izz = 0.075462;
        Ixy = 0;
        Ixz = 0;
        Iyz = 0;
        #Massa
        m = 4.36;
        #Matrizes de Massa e Amortecimento
        M = np.array([[m-Xu_p, 0, 0, 0, 0, 0],[0, m-Yv_p, 0, 0, 0, 0],[0, 0, m-Zw_p, 0, 0, 0],[0, 0, 0, Ixx-
Kp_p, 0, 0],[0, 0, 0, 0, Iyy-Mq_p, 0],[0, 0, 0, 0, 0, Izz-Nr_p]])
        M = np.diag(M)
        D = np.array([[Xu, 0, 0, 0, 0, 0],[0, Yv, 0, 0, 0, 0],[0, 0, Zw, 0, 0, 0],[0, 0, 0, Kp, 0, 0],[0, 0, 0, 0,
Mq, 0],[0, 0, 0, 0, 0, Nr]])
        D = np.diag(D)

        #Horizontal
        #Distância dos motores em relação a y
        D1 = 0.2;
        D2 = -0.2;
        ## Modelo Físico do Submarino
        A = np.array([[Xu/(m-Xu_p), 0],[0, Nr/(Izz-Nr_p)]])

   if avanco:
```

```
    B = np.array([[1/(m-Xu_p)],[-D1/(Izz-Nr_p)]])
  else:
    B = np.array([[1/(m-Xu_p)],[-D2/(Izz-Nr_p)]])

        C = np.array([1,0])

        ## define main variables
        ## Onde aparecer Quail entenda-se Submarino e pra Ninja entenda-se Sensor
    ## Acertar dados de ruído a partir de tentativa e erro com observação do comportamento
prático
        u = entrada; # define acceleration magnitude
        Q = np.array([[0],[0]]) #estado inicial do submarino
        Q_estimate = Q;  #x_estimate of initial location estimation of where the Quail is (what we are
updating)
        QuailAccel_noise_mag = 0.05; #Confiabilidade de quão bons são os motores
        NinjaVision_noise_mag = 0.1;  #pelo o que entendi do código original significa para o
submarino quão boas são as medidas dos sensores
        Ez = NinjaVision_noise_mag^2;# Ez convert the measurement noise (stdv) into covariance
matrix
        Btransposta = B.transpose()
    Ex = QuailAccel_noise_mag^2 * (B*Btransposta); #verificar funcionamento dessa inversa...
        P = Ex; # estimate of initial Quail position variance (covariance matrix)


    # Dado que na realidade não conhecemos, vem junto ao sistema
    #QuailAccel_noise = QuailAccel_noise_mag * parteRandomica;
    #Q= A * Q+ B * u + QuailAccel_noise;
    Q= A * Q+ B * u;
    # Outro dado que na realidade não conhecemos, vem junto à medida
    #NinjaVision_noise = NinjaVision_noise_mag * random();
    #y = C * Q+ NinjaVision_noise;
    y = C * Q;
    Q_loc = np.array([[Q_loc],[Q(1)]]);
    Q_loc_meas = np.array([[Q_loc_meas],[y]]);
    vel = np.array([[vel],[Q(2)]]);


        ## Do kalman filtering

        Q= np.array([[0],[0]]); # re-initized state

        # Predict next state of the quail with the last state and predicted motion.
        Q_estimate = A * Q_estimate + B * u;
        Atransposta = A.transpose();
        P = A * P * Atransposta + Ex;
        Ctransposto = C.transpose()
        inversa = inv(C*P*Ctransposto + Ez)
        K = P*Ctransposto*inversa;
        Q_estimate = Q_estimate + K * (Q_loc_meas(t) - C * Q_estimate);
        Q_loc_estimate = np.array([[Q_loc_estimate],[Q_estimate(1)]]);
        return Q_loc_estimate
```

```
############# MAIN

   # Global variables
vx =              0
vy =              0
viraDireta =      0
viraEsquerda =    0
chanel1 =         17
chanel2 =         27
chanel3 =         22
chanel4 =         23


######### JOYSTICK
#Define as saidas da RbP
#GPIO.setup(11, GPIO.OUT) #Motor pra frente
GPIO.setup(chanel1, GPIO.OUT) #Motor da esquerda
GPIO.setup(chanel2, GPIO.OUT) #Motor da direita
GPIO.setup(chanel3, GPIO.OUT) #Lanterna1
GPIO.setup(chanel4, GPIO.OUT) #Lanterna2

##GPIO.output(11, 0)
GPIO.output(chanel1, 0)
GPIO.output(chanel2, 0)
GPIO.output(chanel3, 0)
GPIO.output(chanel4, 0)

# Inicializações dos conjuntos ESC+motores
mymotor = motor('motores', chanel1, chanel2, simulation=False)

print('***Desligue a energia da ESC')
print('***depois pressione ENTER')
res = raw_input()
mymotor.start()
mymotor.setW(100, 100)

#NOTA: a velocidade angular do motor varia de 0(min) a 100(máx)
#a transformação para PWM é feita na classe "motor"
print('***Ligue a energia da ESC')
print('***Espere o beep-beep')

print('***entao pressione ENTER')
res = raw_input()
mymotor.setW(0, 0)
print('***(Futuro)Espere N beep para as celulas de bateria')
print('***Espere o beeeeeep indicando pronto')
print('***entao pressione ENTER')
res = raw_input()

while 1:

   #Inicia Variáveis de controle
```

```
viraDireta =       0
viraEsquerda =     0

#Inicia Joystick

win = joystick()
win.connect("delete-event", Gtk.main_quit)
win.show_all()
Gtk.main()

####SENSOR

# or bus = smbus.SMBus(0) para placas Revision 1
#bus = smbus.SMBus(1)

#address = 0x68  # Endereco lido para os Acelerometro e Giroscopio pelo i2cdetect

#bus.write_byte_data(address, power_mgmt_1, 0)

#gyro_xout = read_word_2c(0x43)
#gyro_yout = read_word_2c(0x45)
#gyro_zout = read_word_2c(0x47)


#accel_xout = read_word_2c(0x3b)
#accel_yout = read_word_2c(0x3d)
#accel_zout = read_word_2c(0x3f)

accel_xout_scaled = accel_xout / 16384.0
accel_yout_scaled = accel_yout / 16384.0
#accel_zout_scaled = accel_zout / 16384.0

#sensor = BMP085.BMP085()

leituraAvanco = accel_xout_scaled
leituraGuinada = accel_yout_scaled

# Transformação das leituras

leituraAvanco = vx + 1.42857e-6*leituraAvanco
leituraGuinada = vy + 1.42857e-6*leituraGuinada

vx = leituraAvanco
vy = leituraGuinada

# Passa os dados dos sensores pelo Filtro de Kalman

avancoSensorKalman = filtroKalman(1, vx)
guinadaSensorKalman = filtroKalman(0, vy)

#Controle das saídas
```

```
avancoJoystick = 0
if viraDireita && viraEsquerda:
   avancoJoystick = 0.5

if viraEsquerda:
   guinadaJoystick = 0.5
else:
   guinadaJoystick = 0

controleavanco(chanel1, chanel2, avancoJoystick, avancoSensorKalman)
controleGuinada(chanel1, chanel2, guinadaJoystick, guinadaSensorKalman)
```

14 ANEXOS

A seguir estão todos os anexos referentes a este trabalho, e são de suma importância para os interessados a se aprofundar nas programações feitas, principalmente. Apenas os arquivos para testes foram colocados, pois as bibliotecas completas são extensas demais e podem ser facilmente encontradas na internet.

14.1 TESTE DOS DO SENSOR GY-80

```
# -*- coding: utf-8 -*-
import smbus
import math
import Adafruit_BMP.BMP085 as BMP085

# Registradores de gerenciamento de energia

power_mgmt_1 = 0x6b

power_mgmt_2 = 0x6c

def read_byte(adr):

    return bus.read_byte_data(address, adr)


def read_word(adr):

    high = bus.read_byte_data(address, adr)

    low = bus.read_byte_data(address, adr+1)

    val = (high << 8) + low

    return val
```

```
def read_word_2c(adr):
    val = read_word(adr)
    if (val >= 0x8000):
        return -((65535 - val) + 1)
    else:
        return val


def dist(a,b):
    return math.sqrt((a*a)+(b*b))


def get_y_rotation(x,y,z):
    radians = math.atan2(x, dist(y,z))
    return -math.degrees(radians)


def get_x_rotation(x,y,z):
    radians = math.atan2(y, dist(x,z))
    return math.degrees(radians)


# or bus = smbus.SMBus(0) para placas Revision 1
bus = smbus.SMBus(1)

address = 0x68  # Endereco lido para os Acelerometro e Giroscopio pelo i2cdetect

class Sensor():
    while not(raw_input()) :
        bus.write_byte_data(address, power_mgmt_1, 0)
        print "Giroscopio"
        print "---------"
```

```
        gyro_xout = read_word_2c(0x43)
        gyro_yout = read_word_2c(0x45)
        gyro_zout = read_word_2c(0x47)

        print "gyro_xout: ", gyro_xout, " scaled: ", (gyro_xout / 131)
        print "gyro_yout: ", gyro_yout, " scaled: ", (gyro_yout / 131)
        print "gyro_zout: ", gyro_zout, " scaled: ", (gyro_zout / 131)

        print
        print "Acelerometro"
        print "------------------"

        accel_xout = read_word_2c(0x3b)
        accel_yout = read_word_2c(0x3d)
        accel_zout = read_word_2c(0x3f)

        accel_xout_scaled = accel_xout / 16384.0
        accel_yout_scaled = accel_yout / 16384.0
        accel_zout_scaled = accel_zout / 16384.0

        print "accel_xout: ", accel_xout, " scaled: ", accel_xout_scaled
        print "accel_yout: ", accel_yout, " scaled: ", accel_yout_scaled
        print "accel_zout: ", accel_zout, " scaled: ", accel_zout_scaled

        print
        print "Barometro"
        print "------------------"
        sensor = BMP085.BMP085()
        print 'Temp = {0:0.2f} *C'.format(sensor.read_temperature())
        print 'Pressure = {0:0.2f} Pa'.format(sensor.read_pressure())
        print 'Altitude = {0:0.2f} m'.format(sensor.read_altitude())
        print 'Sealevel Pressure = {0:0.2f}
Pa'.format(sensor.read_sealevel_pressure())
```

14.2 TESTE DO MOTOR

O programa que descreve a classe motor é o disposto a seguir.

```
class motor2(object):
   """Manages the currect Angular rotation
   Implements the IO interface using the RPIO lib
   __init_(self, name, pin, kv=1000, RPMMin=1, RPMMax=100, debug=True,
simulation=True):
   More info on RPIO in http://pythonhosted.org/RPIO/index.html"""
```

```
    def __init__(self, name, pin1, pin2, kv=1000, WMin=0, WMax=100,
debug=True, simulation=True):
        self.name = name
        self.powered = False
        self.simulation = simulation
        self.__pin1 = pin1
        self.__pin2 = pin2
        self.__kv = kv
        self.setWLimits(WMin, WMax)
        self.setDebug(debug)

        self.__W1 = self.__WMin
        self.__W2 = self.__WMin
        self.__Wh = 10

        try:
            from RPIO import PWM
            self.__IO = PWM.Servo()
        except ImportError:
            self.simulation = True

    def setDebug(self, debug):
        self.__debug = debug

    def getDebug(self):
        return self.__debug

    def setPin(self, pin1, pin2):
        "set the pin for each motor"
        self.__pin1 = pin1
        self.__pin2 = pin2

    def setKv(self, kv):
        "set the kv for each motor"
        self.__kv = kv

    def setWLimits(self, WMin, WMax):
        "set the pin for each motor"
        if WMin < 0:
            WMin = 0
        self.__WMin = WMin
        if WMax > 100:
            WMax = 100
        self.__WMax = WMax

    def saveWh(self):
        "Save Wh = current W%"
        self.__Wh = self.__W1

    def setWh(self):
```

```
    "Sets current W% =Wh"
    self.__W1 = self.__Wh
    self.__W2 = self.__Wh
    self.setW(self.__W1, self.__W2)

def getWh(self):
    "returns current W% =Wh"
    return self.__Wh

def start(self):
    "Run the procedure to init the PWM"
    if not self.simulation:
        try:
            from RPIO import PWM
            self.__IO = PWM.Servo()
            self.powered = True
            #TODO Decidir como gerenciar WMax < 100
            #para manter o throttle no intervalo 0-100
        except ImportError:
            self.simulation = True
            self.powered = False

def stop(self):
    "Stop PWM signal"

    self.setW(0, 0)
    if self.powered:
        self.__IO.stop_servo(self.__pin1)
        self.__IO.stop_servo(self.__pin2)
        self.powered = False

def increaseW(self, step=1):
    "increases W% for the motor"

    self.__W1 = self.__W1 + step
    self.__W2 = self.__W2 + step
    self.setW(self.__W1, self.__W2)

def decreaseW(self, step=1):
    "decreases W% for the motor"

    self.__W1 = self.__W1 - step
    self.__W2 = self.__W2 - step
    self.setW(self.__W1, self.__W2)

def getW1(self):
    "retuns current W1%"
    return self.__W1

def getW2(self):
```

```
        "retuns current W2%"
        return self.__W2

    def setW(self, W1, W2):
        "Checks W% is between limits than sets it"

        PW1 = 0
        PW2 = 0
        self.__W1 = W1
        self.__W2 = W2
        if self.__W1 < self.__WMin:
            self.__W1 = self.__WMin
        if self.__W1 > self.__WMax:
            self.__W1 = self.__WMax
        if self.__W2 < self.__WMin:
            self.__W2 = self.__WMin
        if self.__W2 > self.__WMax:
            self.__W2 = self.__WMax

        PW1 = (1000 + (self.__W1) * 10)
        PW2 = (1000 + (self.__W2) * 10)
        # Set servo to xxx us
        if self.powered:
            self.__IO.set_servo(self.__pin1, PW1)
            self.__IO.set_servo(self.__pin2, PW2)
```

A seguir temos o código de execução do teste do motor.

```
#solenero.tech@gmail.com
#solenerotech.wordpress.com

#solenerotech 2013.09.06

from motor2 import motor2

mymotor = motor2('motores', 17, 27, simulation=False)
#where 17 is  GPIO17 = pin 11
#where 27 is  GPIO27 = pin 13

print('***Disconnect ESC power')
print('***then press ENTER')
res = raw_input()
mymotor.start()
mymotor.setW(100, 100)

#NOTE:the angular motor speed W can vary from 0 (min) to 100 (max)
#the scaling to pwm is done inside motor class
print('***Connect ESC Power')
print('***Wait beep-beep')
```

```
print('***then press ENTER')
res = raw_input()
mymotor.setW(0, 0)
print('***Wait N beep for battery cell')
print('***Wait beeeeeep for ready')
print('***then press ENTER')
res = raw_input()
print ('increase > a | decrease > z | save Wh > n | set Wh > h|quit > 9')

cycling = True
try:
    while cycling:
        res = raw_input()
        if res == 'a':
            mymotor.increaseW()
        if res == 'z':
            mymotor.decreaseW()
        if res == 'n':
            mymotor.saveWh()
        if res == 'h':
            mymotor.setWh()
        if res == '9':
            cycling = False
finally:
    # shut down cleanly
    mymotor.stop()
    print ("well done!")
```