

DENIS DI FAZIO FRANCOBANDIERA

**DESENVOLVIMENTO DE UM SISTEMA DE
COMUNICAÇÃO SEM FIO EM TEMPO REAL
COM
O CONTROLADOR DE VOO DE UM
MULTIRROTOR**

São Carlos

2017

DENIS DI FAZIO FRANCOBANDIERA

**DESENVOLVIMENTO DE UM SISTEMA DE
COMUNICAÇÃO SEM FIO EM TEMPO REAL COM
O CONTROLADOR DE VOO DE UM MULTIRROTOR**

Trabalho de Conclusão de Curso apresentado à
Escola de Engenharia de São Carlos como parte
dos requisitos para graduação em Engenharia
Mecatrônica

UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS
DEPARTAMENTO DE ENGENHARIA MECATRÔNICA

Orientador: Prof. Eduardo do Valle Simões

São Carlos

2017

AUTORIZO A REPRODUÇÃO TOTAL OU PARCIAL DESTE TRABALHO,
POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS
DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

F814d Francabandiera, Denis Di Fazio
Desenvolvimento de um sistema de comunicação sem
fio em tempo real com um controlador de voo de um
multirroto / Denis Di Fazio Francabandiera; orientador
Eduardo do Valle Simões. São Carlos, 2017.

Monografia (Graduação em Engenharia Mecatrônica) --
Escola de Engenharia de São Carlos da Universidade de
São Paulo, 2017.

1. SARP. 2. Multirroto. 3. Controlador de voo. 4.
Comunicação sem fio. I. Título.

FOLHA DE AVALIAÇÃO

Candidato: Denis Di Fazio Francabandiera

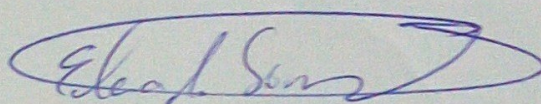
Título: Desenvolvimento de um sistema de comunicação sem fio em tempo real com o controlador de voo de um multirrotor

Trabalho de Conclusão de Curso apresentado à
Escola de Engenharia de São Carlos da
Universidade de São Paulo
Curso de Engenharia Mecatrônica.

BANCA EXAMINADORA

Professor Doutor Eduardo do Valle Simões (Orientador)

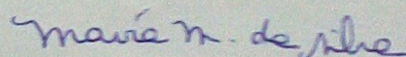
Nota atribuída: 10,0 (DEZ)



(assinatura)

Professora Doutora Máira Martins da Silva

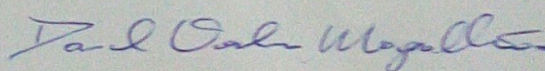
Nota atribuída: 10,0 (DEZ)



(assinatura)

Professor Doutor Daniel Varela Magalhães

Nota atribuída: 10,0 (DEZ)



(assinatura)

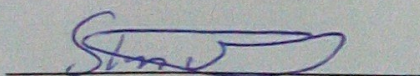
Média: 10,0 (DEZ)

Resultado: APROVADO

Data: 30/06/2017

Este trabalho tem condições de ser hospedado no Portal Digital da Biblioteca da EESC

SIM ☒ NÃO ☐ Visto do orientador



AGRADECIMENTOS

A minha companheira, Miriã, que esteve ao meu lado sempre me motivando.

A meus pais, que desde sempre me incentivaram aos estudos e a seguir em frente.

A Universidade de São Paulo pelos conhecimentos adquiridos ao longo desses anos de formação.

Ao meu orientador, Eduardo Simões, pelos conselhos, aventuras e orientações que me levaram a finalização deste trabalho.

FRANCABANDIERA, Denis Di Fazio **Desenvolvimento de um sistema de comunicação sem fio em tempo real com o controlador de voo de um multirroto**r. São Carlos, 2016. 95f. Trabalho de Conclusão de Curso - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2016.

RESUMO

Nos últimos anos a tecnologia de Sistemas de Aeronaves Remotamente Pilotadas (SARPs) tem se mostrado progressivamente mais presente em iniciativas inovadoras para detecção, análise e monitoramento, tanto no campo militar como em áreas ligadas a economia, como agricultura e pecuária. Grande parte destas novas aeronaves, especificamente as de pequeno porte, são multirrotores: aeronaves de asas rotativas com dois ou mais motores. Usualmente, quando se tratando de um modelo de um aeroplano ou barco, o piloto tem precisão de controle sobre o motor, no qual um aumento no acelerador se traduz em um aumento proporcional de RPM. A diferença de multirrotores é o fato de ser muito difícil de um ser humano ser capaz de controlar a velocidade de rotação de três ou mais motores simultaneamente com precisão suficiente para estabilizar a aeronave no ar. É nesta parte que se torna desejável a presença de um controlador de voo para estabilizar a aeronave. Este é constituído de uma placa de circuito impresso, contendo um processador e dispositivos eletrônicos como sensores e transmissores, e um software de complexidade variável. Sua função é controlar a rotação de cada motor em resposta aos comandos solicitados pelo usuário, em um esquema do tipo “fly-by-wire”. Este trabalho tem como objetivo estudar e alterar o software do controlador ArduPilot de maneira a possibilitar que um sistema de comunicação sem fio em tempo real possa operar entre uma estação de solo e um multirroto

O sistema será formado por um hexacóptero, um controlador de voo fixado na própria aeronave e um módulo de telemetria conectado em ambos os sistemas (multirroto e base de solo). O controlador de voo será baseado em uma placa ArduPilot e se comunicará através do módulo de telemetria de frequência 433 MHz e com capacidade de transmitir dados até 250 kbps. Será desenvolvido também um software de controle que deve ser executado na estação de base e ser capaz de comandar remotamente a aeronave, transmitindo novas rotas de navegação e missões para o robô. Esse trabalho deve viabilizar uma interface de controle remoto que deverá possibilitar trabalhos futuros utilizando técnicas de inteligência artificial como inteligência de enxames e algoritmos evolutivos para controlar sistemas multirro

Palavras-chave: SARP. Multirroto. Comunicação sem fio. Controlador de voo.

FRANCABANDIERA, Denis Di Fazio. **Design of a real time wireless communication system using the flight controller of a multirotor.** São Carlos, 2016. 95f. Course Final Project - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2016.

ABSTRACT

In recent years the technology of Remotely Piloted Aircraft System (RPAS) has been progressively more present in innovative initiatives for detection, analysis and monitoring, both in the military field and in areas related to economy, such as agriculture and livestock. Most of these new aircrafts, especially small ones, are multirotors: rotary-wing aircraft with two or more engines. Usually, in the case of a model of an airplane or ship, the pilot has control accuracy of the engine, in which an increase in the accelerator is translated into a proportional increase of RPM. The difference in multirotors is that it is very difficult for a human to be able to control the rotation speed of three or more motors simultaneously with sufficient precision to stabilize the aircraft in the air. Here is where it is desirable the presence of a flight controller to stabilize the aircraft. The flight controller consists of a printed circuit board containing a processor and electronic devices such as sensors and transmitters and a variable complexity software. Its function is to control the rotation of each motor in response to commands requested by the user in a scheme fly-by-wire. This work aims to study and change the ArduPilot driver software in order to enable a wireless communication system in real time between a ground station and multirotor. The system is formed by a hexacopter, a flight controller fixed on the aircraft itself and a telemetry module connected on both systems (multirotor and ground station). The flight controller is based on an ArduPilot card and it will communicate via the telemetry module, which its frequency is 433 MHz and it is capable of transmitting data up to 250 kbps. It will also be developed a control software, which it will be executed on the ground station and it will be able to remotely control the aircraft by passing new shipping routes and missions to the robot. This work should enable a remote control interface that should enable future work using artificial intelligence techniques as swarm intelligence and evolutionary algorithms to control multirobot systems.

Keywords: RPAS. Multirotor. Wireless communication. Flight controller.

LISTA DE ILUSTRAÇÕES

Figura 1 – Ilustração de uma aeronave remotamente pilotada com quatro motores (flickr.com, 2014)	21
Figura 2 – ArduPilot Mega, controlador de voo com vários sensores de medição inercial (RCGroups.net)	26
Figura 3 – Ilustração dos movimentos de um multirrotor	27
Figura 4 – Estação em solo preparada para a captação de dados provenientes de um VANT (aerojeep.com)	28
Figura 5 – Mission Planner, aplicativo de estação em solo de código aberto (ardupilot.org)	30
Figura 6 – Visual Studio, ambiente de desenvolvimento integrado desenvolvido pela Microsoft (visualstudio.com)	33
Figura 7 – Git, sistema de controle de versão distribuído usado para desenvolvimento de software (git-scm.com)	34
Figura 8 – GitHub, plataforma de desenvolvimento de projetos de código aberto (github.com)	35
Figura 9 – Sistema de comunicação usual entre um controle radiotransmissor, um VANT e uma estação em solo	37
Figura 10 – Sistema de comunicação proposto pelo trabalho	38
Figura 11 – Estação em solo contendo o software Mission Planner	39
Figura 12 – Instalação do Mission Planner	40
Figura 13 – Instalação do Visual Studio	40
Figura 14 – Instalação do DirectX	41
Figura 15 – GitHub Desktop	41
Figura 16 – Adquirindo repositório	42
Figura 17 – Clonando repositório	42
Figura 18 – Abrindo repositório diretamente do aplicativo GitHub Desktop	43
Figura 19 – Modificando o tipo de projeto	43
Figura 20 – Desabilitando Sign the ClickOnce manifests	44
Figura 21 – Adicionando bibliotecas externas	44
Figura 22 – Compilando projeto	45
Figura 23 – Janela Batch Build	45
Figura 24 – Inicializando o projeto	45
Figura 25 – a) Criando um novo branch	46
Figura 26 – b) Criando um novo branch	46
Figura 27 – c) Criando um novo branch	47
Figura 28 – Seleccionando o novo branch	47
Figura 29 – Tela escolhida para inserir o novo método	48

Figura 30 – Modelo da nova tela	48
Figura 31 – Tela Planner vista do código-fonte	49
Figura 32 – Aumentando altura da tela	49
Figura 33 – Inserindo o componente label na tela	50
Figura 34 – Inserindo o botão na tela	50
Figura 35 – Inserindo nova classe ao projeto	51
Figura 36 – Tela de configurações do teclado	52
Figura 37 – Tela de debug da IDE com as teclas pressionadas	53
Figura 38 – Mission Planner rodando uma simulação de multirroto	55
Figura 39 – Antena construída manualmente	56
Figura 40 – Perda de sinal com as novas antenas em uma distância de aproximadamente 160m	57
Figura 41 – Distância percorrida	57
Figura 42 – Modem de telemetria utilizado no projeto conectado a um smartphone . . .	62

LISTA DE TABELAS

Tabela 1	–	Comparação entre controladores de voo	26
Tabela 2	–	Comparação entre estações em solo	29

LISTA DE ABREVIATURAS E SIGLAS

ARP	Aeronave Remotamente Pilotada
VANT	Veículo Aéreo Não-Tripulado
CV	Controlador de voo
GPS	Global Positioning System (Sistema de Posicionamento Global)
PID	Proporcional Integral Derivativo
APM	ArduPilot Mega
SO	Sistema Operacional
OSI	Open Source Initiative
CLR	Common Language Runtime
ANATEL	Agência Nacional de Telecomunicações
IDE	Integrated Development Environment
SITL	Software in the Loop
PID	Controlador Proporcional Integral Derivativo
ECMA	European Computer Manufacturers Association
CLR	Common Language Runtime
RTL	Return-to-Launch
MAVLink	Micro Air Vehicle Link

SUMÁRIO

1	INTRODUÇÃO	21
1.1	CONTEXTO	21
1.2	OBJETIVOS	22
2	REVISÃO BIBLIOGRÁFICA	25
2.1	CONTROLADORES DE VOO	25
2.1.1	MODELOS DE CONTROLADORES DE VOO	26
2.1.2	ArduPilot Mega	26
2.2	MOVIMENTOS DE UM VANT	27
2.3	ESTAÇÕES EM SOLO	28
2.3.1	MODELOS DE ESTAÇÕES EM SOLO	29
2.3.2	Mission Planner	30
2.4	PROTOCOLO MAVLINK	31
2.5	CÓDIGO ABERTO	31
2.6	MICROSOFT VISUAL STUDIO	33
2.7	GIT	34
2.7.1	GitHub	35
3	DESENVOLVIMENTO DA INTERFACE DE COMUNICAÇÃO ENTRE ESTAÇÃO EM SOLO E AERONAVE	37
3.1	Diagrama do sistema de comunicação	37
3.2	Configurando o software Mission Planner	39
3.2.1	Descarregando e instalando o Mission Planner	39
3.2.2	Trabalhando com o código-fonte do Mission Planner	41
3.3	Modificando o software Mission Planner	46
3.3.1	Adicionando um novo branch ao projeto do Mission Planner	46
3.3.2	Usando o novo branch para modificar o código-fonte	48
3.3.2.1	Inserindo o botão de configurações do teclado	49
3.3.2.2	Inserindo a interface de configurações do teclado	51
3.3.2.3	Criando a classe Keyboard	52
3.3.2.4	Criando comunicação entre teclado e VANT	53
3.3.3	Testando a nova implementação do software	54
3.4	Aprimorando os dispositivos de localização e comunicação	55
3.5	Testes de Confiabilidade	57
3.5.1	Testes com as novas antenas	57

3.5.2	Testes de Failsafe	58
3.5.3	Teste final em campo aberto	59
4	CONSIDERAÇÕES FINAIS	61
4.1	CONCLUSÃO	61
4.2	CONTRIBUIÇÕES	61
4.3	TRABALHOS FUTUROS	62
	REFERÊNCIAS	63
	ANEXO A – CÓDIGO-FONTE KEYBOARDSETUP.CS	67
	ANEXO B – CÓDIGO-FONTE KEYBOARD.CS	77
	ANEXO C – CÓDIGO-FONTE THREAD DO TECLADO	93

1 INTRODUÇÃO

1.1 CONTEXTO

Veículos Aéreos Não-Tripulados (VANTs), popularmente conhecidos como drones ou também ARPs (Aeronaves Remotamente Pilotadas) são aeronaves capazes de voarem através do controle de um operador que não está a bordo. Comumente associados ao uso militar, tem-se visto um interesse crescente no desenvolvimento de VANTs para o uso doméstico, setores privados e recreacional. Isto se deve a diminuição de custo ao longo dos anos da tecnologia VANT e também ao fato destes possuírem vantagens funcionais sobre os veículos tripulados. VANTs podem pairar no ar, além de manobrar suavemente e precisamente através de pequenos espaços ou em conjunto com outros drones. Tudo isso enquanto carregando instrumentos como uma câmera de vídeo estabilizada e uma infinidade de outras tecnologias a bordo. A extensão de sua versatilidade é o que os torna uma opção viável para uma série de tarefas diferentes (CAVOUKIAN, 2012).



Figura 1 – Ilustração de uma aeronave remotamente pilotada com quatro motores (flickr.com, 2014)

O uso de VANTs tem sido de grande utilidade em diversas áreas tais como transporte de medicamentos e suprimentos às vítimas de catástrofes (DOHERTY; RUDOL, 2007), coleta de dados atmosféricos (GONÇALVES et al., 2006), pecuária de precisão (CARDOSO et al., 2016) e uso militar (RAMOS, 2014). Isto demonstra a grande gama de aplicações atuais dos drones e os interesses econômicos envolvidos em relação ao seu desenvolvimento e aplicação.

Conforme Neris (2001), o voo autônomo, uma das principais características dessas aeronaves, permite uma redução de custos quando comparados aos veículos aéreos convencionais. Essa diminuição ocorre, principalmente, pela inexistência de piloto a bordo da aeronave. Entretanto, ainda há os gastos com todos os equipamentos e montagem. No caso de um VANT, este é dividido em: estrutura, motores, hélices, controladores de velocidade, controlador de voo, baterias, sensores e rádio. Conforme Demolinari (2016), um VANT robusto e com plataforma generalista similar a utilizada neste projeto, tem um custo de R\$2539,81. Já um rádiocontrole de quatro canais tem um custo médio de R\$450,00. Sendo, portanto, aproximadamente 18% do custo de construção de um VANT. O que o torna o componente com o maior preço unitário do projeto.

O uso do rádiocontrole em VANTs provém de métodos utilizados antes na área de aeromodelos. Neste meio, o piloto necessita possuir total controle de seu veículo, a comunicação entre piloto e máquina deve ser instantânea e fina, para que assim, o menor dos movimentos feitos pelo piloto seja rapidamente detectado pelo equipamento. Por essa razão, rádiocontroles costumam ser de alto custo (BODDINGTON, 2004).

Por outro lado, no âmbito de sistemas VANTs, por mais experiente que seja o piloto, o controle de um helicóptero é desafiante e mentalmente cansativo. Isto se deve, em parte, às instabilidades inerentes e altamente não-lineares da natureza de um helicóptero. Entradas de controle corretivas constantes são necessárias para manter uma trajetória de voo desejada na qual coloca uma grande carga sobre o piloto humano (MONTGOMERY; BEKEY, 1998). Para isso, torna-se necessária a utilização de controladores de voo, no qual hardware e software atuam em conjunto vários ciclos por segundo para manter a aeronave em equilíbrio no ar. Conforme a tecnologia avança, é necessário cada vez menos o uso de rádiocontroles robustos, pois os ajustes finos ficam reservados ao controlador. Além disso, atualmente as missões que operam com VANTs são normalmente atividades de mapeamento (SOUZA, 2015), patrulha (GIRARD; HOWELL; HEDRICK, 2004) ou varredura de setores (CASBEER et al., 2006), realizadas normalmente sob controle de um piloto automático de forma autônoma, que, caso necessitem de trabalho humano, são para movimentos simples, linhas retas que devem ser seguidas pelo veículo.

Tendo em vista a redução de custo de um projeto de VANT, este trabalho tem como tese comprovar a possibilidade de se remover por completo a interface de alto custo por meio de um rádiocontrole e substituí-lo por uma outra, de menor custo e simplificada, porém de alta confiabilidade. Retirando-se o uso da interface antiga é possível diminuir até pela metade o preço da construção de uma aeronave, pois a nova interface pode ser aplicada em um componente já presente, a estação em solo constituída de um notebook comum. Assim, com a redução de custo do sistema, comunidades que antes não podiam adquirir um VANT, tais como escolas públicas ou mesmo iniciações científicas, poderão construir sua própria aeronave.

1.2 OBJETIVOS

O objetivo deste projeto é desenvolver um novo sistema de controle para VANTs diretamente pelo software que executa a estação em solo, normalmente um computador ou notebook, por meio de um modem sem fio e um teclado, sem necessidade de, um rádiocontrole de aeromodelo. O projeto aborda, especificamente, o uso e alteração do código em linguagem C# de um software livre denominado *Mission Planner*, que constitui uma estação de solo, para fim de estendê-lo atribuindo-o a nova função. O software possui um repositório localizado no site GitHub que tem a contribuição de diversos desenvolvedores ao redor do mundo.

Primeiramente é demonstrado testes de um sistema de comunicação através do uso de

um joystick, para que o operador possa pilotar remotamente a aeronave. Depois é feita a nova implementação que utiliza os teclados do notebook para se comunicar com a aeronave. Ao final deste trabalho, serão levantadas sugestões para possíveis futuros projetos que possam controlar a aeronave a partir de um software controlador de missões executado pela estação de solo.

2 REVISÃO BIBLIOGRÁFICA

2.1 CONTROLADORES DE VOO

Em situações em que um usuário se encontra na obrigação de pilotar um multirroto, se torna necessário o auxílio de um controlador de voo. Um controlador de voo (CV) ou piloto automático (*autopilot* em inglês) é um sistema utilizado para controlar a trajetória de uma aeronave sem controle constante de um operador humano sendo necessário. Os pilotos automáticos não substituem o operador humano, mas os ajudam a controlar o VANT, permitindo que eles se concentrem em aspectos mais amplos da operação, como monitorar a trajetória, o clima e o outros equipamentos da operação (FAA, 2009). Além disso, o CV é composto por uma pequena placa de circuito com complexidade variada, tendo como função determinar a rotação por minuto (RPM) necessária de cada motor em resposta a uma entrada: um comando provindo do piloto para executar um movimento para frente é alimentado ao controlador de voo que determina como manipular adequadamente os motores.

A maioria dos CVs também utilizam sensores para auxiliar seus cálculos. Estes últimos variam desde simples giroscópios para orientação até barômetros para cálculos automáticos de altitude (ARDUPILOT, 2016e). Sistemas de posicionamento global (GPS em inglês) também podem ser usados para piloto automático ou para fins de aterrissagens com segurança (*failsafe*) (ARDUPILOT, 2016b). Dependendo da escolha de controlador de voo, há vários softwares disponíveis para cenários distintos de uso. Com um controlador de voo bem configurado, o comportamento da aeronave deve corresponder aos comandos de entrada inseridos pelo piloto. CVs são configuráveis e programáveis, permitindo ajustes a diferentes tipos de configurações de multirrotores.

Controladores proporcionais integrais derivativos (PIDs) são usados para aumentar a precisão do controlador de voo. Um PID calcula continuamente um "valor de erro" como a diferença entre uma medida da variável de processo e um desejado ponto de ajuste. O controlador tenta minimizar o erro ao longo do tempo por ajuste de uma variável de controle que une as ações proporcional, integral e derivativa (ÅSTRÖM; HÄGGLUND, 1995). Muitos controladores de voo permitem diferentes modos de voo, selecionáveis através do uso de um interruptor transmissor ou via comandos enviados por telemetria. Um exemplo de um sistema de três posições seria um modo manual, um modo de travamento de altitude e piloto automático. Diversas configurações podem ser aplicadas a diferentes tipos de missões.

2.1.1 MODELOS DE CONTROLADORES DE VOO

Existem no mercado vários modelos diferentes de controladores de voo. Abaixo, segue uma tabela com as características mais relevantes dos CVs mais populares atualmente:

Tabela 1 – Comparação entre controladores de voo

	ArduPilot Mega	DJI (Wookong-M Waypoint)	ZeroUAV (YS-X6)	Mikrokopter
Componentes	Piloto Automático, GPS, telemetria	Piloto automático, GPS, telemetria	Piloto automático, GPS, telemetria	Piloto automático, GPS, telemetria
Autonomia completa	Sim	Sim	Sim	Sim
Waypoints (n suportado)	166	50	16	100 em um raio de 250m
Edição em tempo de voo	Sim	Sim	Modo guiado somente	Sim
Comunicação sem fio	Sim (telemetria via rádio)	Sim (bluetooth)	Não	Não
Suporte Geo-Fence	Sim	Não	Não	Não
Simulação robusta (HIL/SIL)	Ambos	HIL somente	Não	SIL somente
Código aberto de protocolo de comunicação	Sim (MAVLink)	Não	Não	Não
Código aberto	Sim	Não	Não	Não

Analisando os dados acima é possível afirmar que o modelo mais propício ao projeto é o controlador ArduPilot Mega (APM), pois é o único que dispõe de código aberto e também é o único que possui uma comunicação sem fio por meio de um modem. Este último requisito é necessário para a implementação da funcionalidade de pilotagem remota proposta pelo trabalho.

2.1.2 ARDUPILOT MEGA

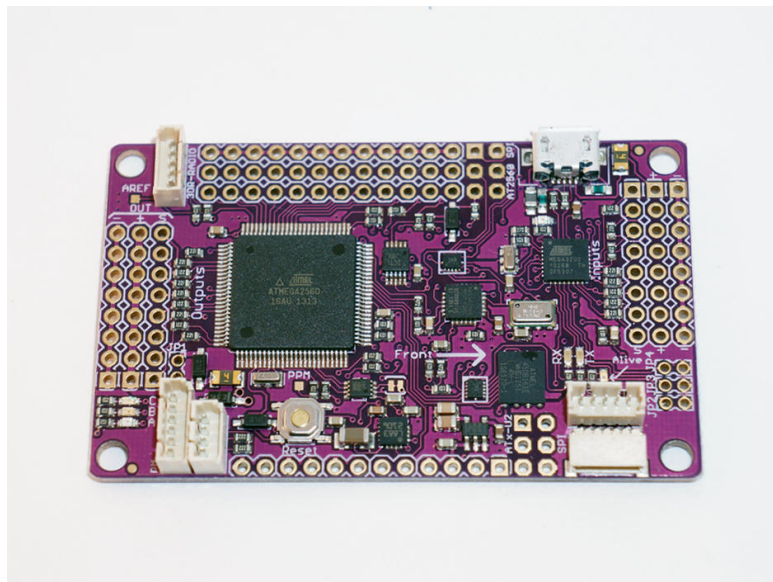


Figura 2 – ArduPilot Mega, controlador de voo com vários sensores de medição inercial (RC-Groups.net)

APM é um controlador de voo totalmente programável através da interface Arduino IDE. A placa é composta por sensores e um módulo GPS capazes de criar um veículo aéreo não pilotado totalmente funcional. O piloto automático controla tanto a estabilidade quanto a navegação da aeronave, eliminando a necessidade de um sistema separado de estabilização. Como visto acima na tabela, ele também suporta um modo “fly-by-wire” capaz de estabilizar o

veículo enquanto controlado manualmente por um controle radiotransmissor. Tanto o hardware quanto o software são código aberto. Este último precisa ser descarregado e instalado na placa pelo próprio usuário. Por fim, quando encomendada, a placa já vem com todos os componentes soldados (APM..., 2016).

2.2 MOVIMENTOS DE UM VANT

O movimento de um VANT é controlado por rotações que ocorrem nos três eixos principais que atravessam um VANT perpendicularmente em seu centro de gravidade: o yaw, o pitch e o roll. Yaw é o desvio ou rotação da frente do VANT para a esquerda ou direita. Já o pitch, é o movimento de ir ou para frente ou para trás. Por último, roll é o movimento para os lados. Além dos movimentos citados acima, é possível também mais dois movimentos: o de subida e o de descida que podem ser feitos de formas distintas em VANTs de tipos diferentes. A Figura 3 ilustra um multirrotor e seus três eixos principais, indicando pela seta vermelha a frente do VANT.

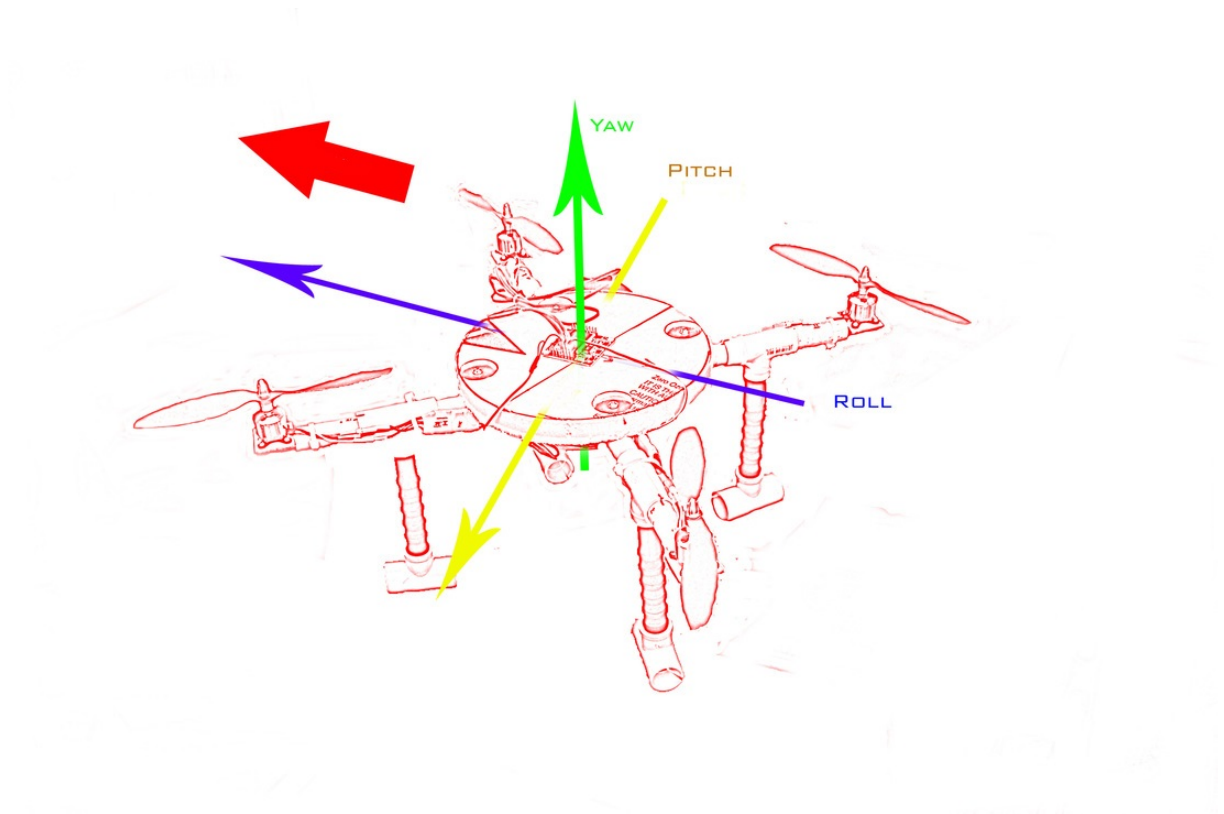


Figura 3 – Ilustração dos movimentos de um multirrotor

2.3 ESTAÇÕES EM SOLO

Uma estação em solo (ground station em inglês) é normalmente um software que é executado em um computador localizado em solo no qual se comunica com o VANT via telemetria sem fio. A estação é capaz de exibir dados em tempo real relacionados ao desempenho e posição do drone, além de servir também como um "cockpit virtual" no qual é possível visualizar muitos dos instrumentos utilizados em uma aeronave de verdade.



Figura 4 – Estação em solo preparada para a captação de dados provenientes de um VANT (aerojeep.com)

2.3.1 MODELOS DE ESTAÇÕES EM SOLO

Segue tabela de comparação das estações em solo mais populares atualmente compatíveis com o controlador de voo escolhido no tópico anterior (APM):

Tabela 2 – Comparação entre estações em solo

	QGroundcontrol	Mission Planner	APM Planner	Tower	AndroPilot **	UGCS	Drone Deploy
Compatível com MAVLink	✓	✓	✓	✓	✓	✓	✓
Código aberto	✓	✓	✓	✓	✓	✗	✗
Compatível com APM	✓	✓	✓	✓	✓	✓	✓
Compatível com PX4	✓	✓	✓	✗	✓	✓	✓
Plataforma							

* mobile
 ** desenvolvimento parado
 *** roda em Mac ou Linux utilizando-se Mono

Fonte: Ground Control Station Benchmark Study (Intel)

Analisando os dados acima, nota-se que para se trabalhar com o código utilizando o sistema operacional (SO) *Windows*, tem-se três opções de escolha: QGroundcontrol, Mission Planner e APM Planner.

- QGroundControl (2017) desenvolvida utilizando a linguagem C++, esta ground station é compatível com APM, porém tem seu software principalmente focado em outro modelo de controlador de voo, o PX4. Seu suporte para APM é algo muito recente, estando na fase beta ainda.
- APM Planner (2016) possui a menor comunidade das três estações de solo escolhidas. Além disso, possui uma menor gama de funcionalidade.
- Mission Planner (2016) contendo a maior comunidade de colaboradores ativos e desenvolvida utilizando como base a linguagem C# para uso em computadores com sistema operacional *Windows*, apesar disso o software também pode ser executado em Mac usando Mono. Além disso, outra característica importante deste software é a possibilidade de se criar pequenos scripts escritos em linguagem Python que estendem as funcionalidades do programa. Esta é a estação escolhida para ser utilizada no projeto.



Figura 5 – Mission Planner, aplicativo de estação em solo de código aberto (ardupilot.org)

2.3.2 MISSION PLANNER

Mission Planner fornece as mais completas funcionalidades para configuração de veículos (aviões, helicópteros e rovers) e também para monitoramento de voo. O software pode ser adquirido de forma gratuita em seu repositório git hospedado no site do GitHub (MISSION PLANNER, 2017).

O aplicativo é capaz dentre outras funções:

- Carregar/atualizar o firmware (APM, PX4) dentro do controlador de voo instalado no drone
- Configurar e calibrar o veículo até atingir um desempenho desejável.
- Planejar, salvar e carregar missões autônomas em seu VANT com waypoints configurados no Google Maps ou em outro mapa de geolocalização.
- Analisar e salvar logs
- Simuladores Software In The Loop (SITL): permitem ao usuário simular um avião, rover (robô terrestre), helicóptero ou multirrotor sem a necessidade da presença de um hardware. Através do SITL, é gerado um executável que permite testar o comportamento do veículo sem a presença física de um hardware (SITL, 2016).
- Através de telemetria é possível:
 - Monitorar o veículo em tempo de operação

- Analisar e salvar logs de telemetria
- Operar o veículo em FPV (first person view)

2.4 PROTOCOLO MAVLINK

Micro Air Vehicle Link (MAVLink) é um protocolo de código aberto (MAVLink, 2017) para comunicação com pequenos veículos não tripulados que foi lançado no início de 2009. O protocolo foi projetado como uma biblioteca de mensagens do tipo *header-only*, isto é, as definições completas de todas as macros, funções e classes que compõem a biblioteca são visíveis para o compilador em um formulário de arquivo de cabeçalho. A vantagem das bibliotecas *header-only* é que estas não precisam ser compiladas separadamente, empacotadas e instaladas para serem usadas. Tudo o que é necessário é apontar o compilador no local dos cabeçalhos. O protocolo é usado principalmente para comunicação entre uma estação em solo e um VANT, e na inter-comunicação do subsistema do veículo. Pode ser usado para transmitir a orientação do veículo, sua localização GPS e velocidade. O mesmo foi extensivamente testado nas plataformas *PX4*, *PIXHAWK*, *APM* e *Parrot AR.Drone* (MEIER, 2009).

2.5 CÓDIGO ABERTO

Código aberto, ou *open source* em inglês, é um modelo de desenvolvimento que promove um licenciamento livre para design, estudo, modificação, distribuição ou esquematização de um produto (LAURENT, 2004). O termo “código aberto” foi popularizado pela Open Source Initiative (OSI, 1998), organização criada para incentivar uma aproximação de entidades comerciais com o software livre. Um adendo importante se diz ao respeito em enfatizar que o open source se difere do software livre por não respeitar as quatro liberdades essenciais definidas pela Free Software Foundation (FSF). Qualquer licença de software livre é também uma licença de software de código aberto, mas o contrário não necessariamente também é verdade (STALLMAN, 2007).

A definição de *open source* foi criada pela OSI e tem como características:

- Distribuição livre:

A licença não deve restringir em hipótese alguma a venda ou distribuição do programa gratuitamente, como componente de outro aplicativo ou não.

- Código-fonte:

O programa deve incluir seu código-fonte e deve permitir a sua distribuição também na forma compilada. Se o software não for distribuído com seu código-fonte, deve haver algum meio de se obter o mesmo seja via rede ou com custo apenas de reprodução. O código deve ser legível e inteligível por qualquer programador.

- Trabalho Derivados:

A licença deve permitir modificações e trabalhos derivados além de conceder que os mesmos sejam distribuídos sobre os termos da licença original.

- Integridade do autor do código-fonte:

A licença pode restringir o código-fonte de ser distribuído em uma forma modificada apenas se a licença permitir a distribuição de arquivos de atualização com o código-fonte para o propósito de modificar o programa no momento de sua construção. A licença deve explicitamente permitir a distribuição do programa construído a partir do código-fonte modificado. Contudo, a licença pode ainda requerer que programas derivados tenham um nome ou número de versão diferentes do programa original.

- Não discriminação contra pessoas ou grupos:

A licença não pode ser discriminatória contra qualquer pessoa ou grupo de pessoas

- Não discriminação contra áreas de atuação:

A licença não deve restringir qualquer pessoa de usar o programa em um ramo específico de atuação. Por exemplo, ela não deve proibir que o programa seja usado em uma empresa, ou de ser usado para pesquisa genética.

- Distribuição da Licença:

Os direitos associados ao programa devem ser aplicáveis para todos aqueles cujo programa é redistribuído, sem a necessidade da execução de uma licença adicional para estas partes.

- Licença não específica a um produto:

Os direitos associados ao programa não devem depender que o programa seja parte de uma distribuição específica de programas. Se o programa é extraído desta distribuição e usado ou distribuído dentro dos termos da licença do programa, todas as partes para quem o programa é redistribuído devem ter os mesmos direitos que aqueles que são garantidos em conjunção com a distribuição de programas original.

- Licença não restrinja outros programas:

A licença não pode colocar restrições em outros programas que são distribuídos juntos com o programa licenciado. Isto é, a licença não pode especificar que todos os programas distribuídos na mesma mídia de armazenamento sejam programas de código aberto.

- Licença neutra em relação a tecnologia:

Nenhuma cláusula da licença pode estabelecer uma tecnologia individual, estilo ou interface a ser aplicada no programa.

2.6 MICROSOFT VISUAL STUDIO

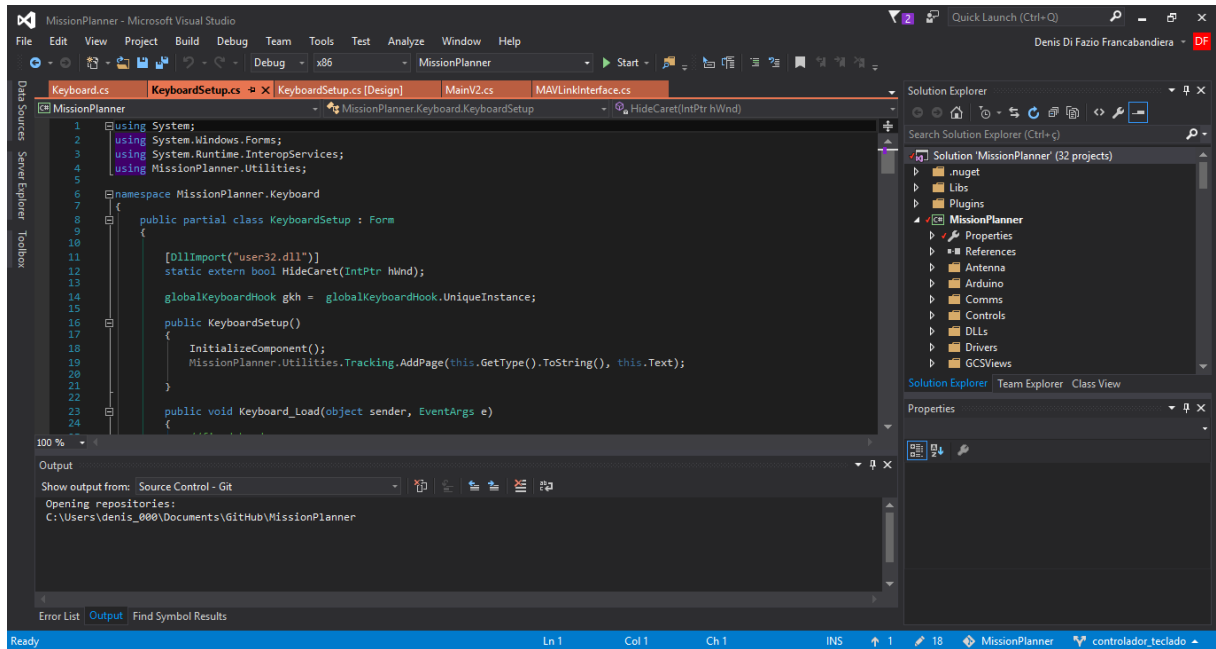


Figura 6 – Visual Studio, ambiente de desenvolvimento integrado desenvolvido pela Microsoft (visualstudio.com)

Visual Studio (2017) é um Ambiente Integrado do Desenvolvimento (IDE em inglês) da *Microsoft*. É usado para desenvolvimento de programas computacionais para Windows como também para criação de *websites*, aplicações web e serviços web. Visual Studio utiliza softwares de desenvolvimento fabricados pela *Microsoft* tais como Windows API, Windows Forms, Windows Presentation Foundation, Windows Store e Microsoft Silverlight. O IDE pode tanto produzir programas nativos como também de código gerenciado (*managed code*), ou seja, softwares que são executados através da máquina virtual Common Language Runtime (CLR).

Visual Studio inclui um editor de código muito completo com refatoração de código e o completador de código desenvolvido pela *Microsoft* chamado *IntelliSense*. O depurador integrado funciona tanto a nível de código-fonte como também a nível de linguagem de máquina. O IDE aceita plug-ins que aumentam as funcionalidades adicionando suporte para sistemas de controle de versionamento tais como Subversion ou GIT e também novas ferramentas para o editor.

Antes da versão 2013, versões comerciais do Visual Studio eram disponibilizadas de graça para estudante através do programa de ensino da Microsoft's DreamSpark. Atualmente, com a versão Visual Studio 2013, a *Microsoft* disponibiliza uma edição gratuita que suporta plugins denominada *Community*.

2.7 GIT



Figura 7 – Git, sistema de controle de versão distribuído usado para desenvolvimento de software (git-scm.com)

GIT (2017) é um sistema de controle de versão distribuído usado para desenvolvimento de software e outras tarefas que exigem versionamento. Tem como ênfase velocidade, integridade de dados e suporte a *workflows* distribuídos, não lineares. Git foi inicialmente projetado e desenvolvido por Linus Torvalds em 2005 para o desenvolvimento do *kernel* Linux, mas foi adotado por muitos outros projetos (GIT, 2005). Atualmente, repositórios Git podem ser hospedados de forma gratuita em vários sites, sendo o mais famoso deles *GitHub*.

Cada diretório de trabalho Git é um repositório com um histórico completo e acompanhamento de revisões, não dependente de acesso a uma rede ou a um servidor central. O Git é um software livre, distribuído sob os termos da versão 2 da GNU (General Public License). Cada vez que uma pessoa salva ou consolida (*commit*) o estado de um projeto no Git, é como se ele tirasse uma foto de todos os seus arquivos naquele momento e armazenasse uma referência para essa captura. Um *commit* no contexto destes sistemas de controle de versão refere-se a submeter as últimas alterações do código-fonte ao repositório e fazer com que estas alterações se tornem parte da versão principal (*head*) do repositório. Deste modo, quando outros usuários fazem uma cópia (*checkout*) do repositório, eles receberão a versão enviada mais recentemente, a menos que eles especifiquem que querem recuperar uma versão anterior do código-fonte no repositório (GIT, 2017).

2.7.1 GITHUB



Figura 8 – GitHub, plataforma de desenvolvimento de projetos de código aberto (github.com)

GitHub (2017a) é um site que teve desenvolvimento iniciado em 1 outubro de 2007. O site foi lançado em 1 de abril de 2008 por Tom Preston-Werner, Chris Wanstrath e PJ Hyett depois de se tornar disponível por um período curto de meses como um teste beta. O software que executa o GitHub foi escrito utilizando Ruby on Rails e Erlang.

Projetos hospedados no GitHub pode sem acessados e manipulados utilizando a interface padrão de linha de comando do Git. Além disso, também há disponível um aplicativo com uma interface gráfica de usuário (GUI em inglês) para uso *desktop*. O site também permite usuários registrados e não-registrados navegarem em seus repositórios públicos. Além disso, também fornece funções sociais tais como *feeds*, seguidores, *wikis* e um gráfico social no qual exibe como os desenvolvedores trabalham em suas versões (*forks*) e qual *fork* é o mais recente. Além disso, um usuário precisa criar uma conta para poder contribuir com conteúdo. Com uma conta registrada usuários podem discutir, administrar, criar repositórios e enviar contribuições para outros repositórios e também revisar códigos (GITHUB, 2017b).

3 DESENVOLVIMENTO DA INTERFACE DE COMUNICAÇÃO ENTRE ESTAÇÃO EM SOLO E AERONAVE

3.1 DIAGRAMA DO SISTEMA DE COMUNICAÇÃO

Um sistema de comunicação comumente utilizado composto por um rádiocontrole, um drone e uma estação em solo pode ser representado conforme mostrado pela Figura 9:

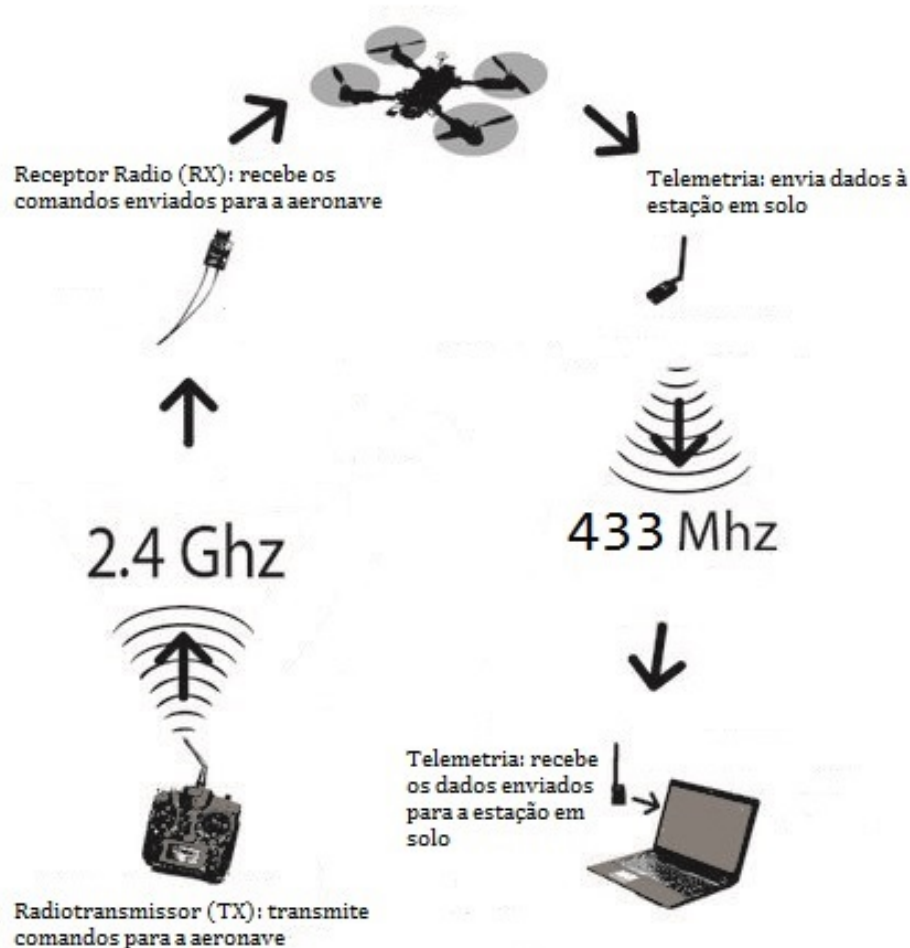


Figura 9 – Sistema de comunicação usual entre um controle radiotransmissor, um VANT e uma estação em solo

Como pode ser visto na Figura 9, o meio de comunicação entre os componentes é unidirecional, no qual o rádiocontrole apenas envia comandos ao VANT e a estação em solo somente recebe os dados do mesmo através de sua telemetria. É importante notar que os dois canais de comunicação, rádiocontrole-drone e drone-estação, devem possuir frequências diferentes, caso contrário, o sistema entrará em conflito e falhará. Além disso, outro aspecto importante da imagem é a frequência utilizada tanto pela telemetria como quanto pelo rádiocontrole, ambas necessitam seguir as faixas de radiofrequências permitidas pela Agência Nacional de Telecomunicações (Anatel) conforme a resolução ANATEL N° 506/2008 Art.8-Tabela I.

Este trabalho propõe remover a dependência de um rádiocontrole e tornar bidirecional a comunicação entre drone e estação em solo, visto na Figura 10 a seguir:

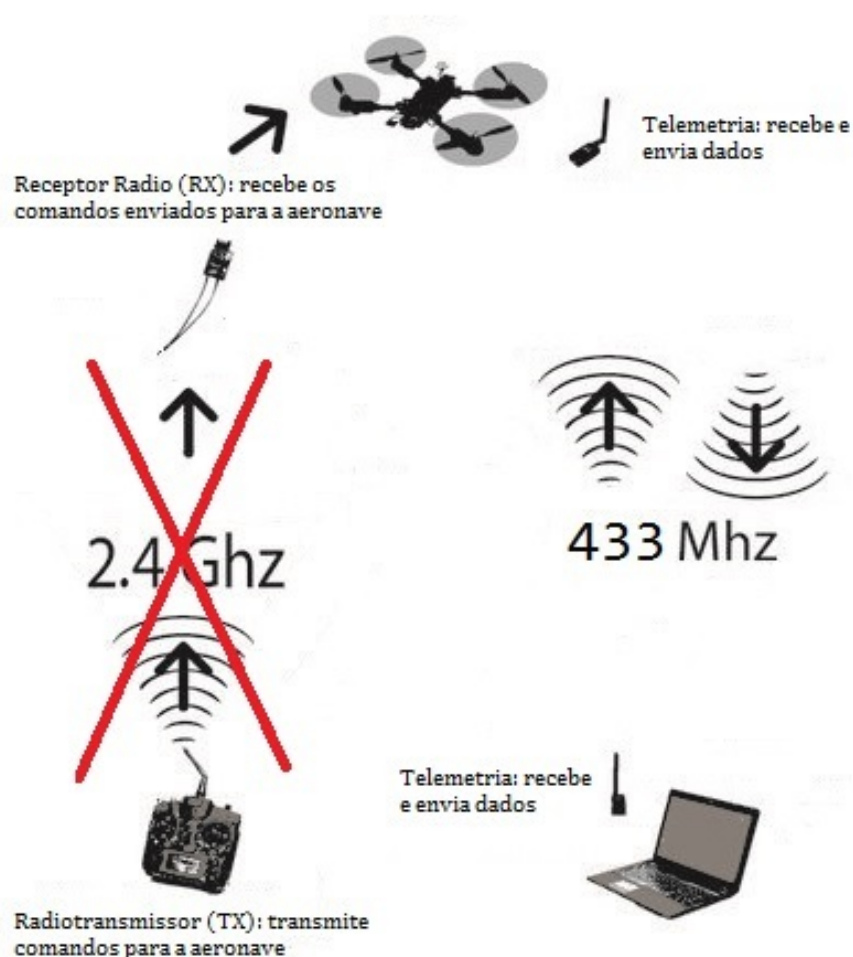


Figura 10 – Sistema de comunicação proposto pelo trabalho

Para tal feito, é necessário modificar o software utilizado pela estação em solo para que o mesmo passe a enviar comandos.

3.2 CONFIGURANDO O SOFTWARE MISSION PLANNER



Figura 11 – Estação em solo contendo o software Mission Planner

Devido ao fato do software *Mission Planner* ser de código aberto, é possível analisar seu modo de funcionamento e também alterá-lo. Para isso, é necessário primeiro instalá-lo em uma estação em solo (notebook) contendo o sistema operacional *Windows*, uma máquina virtual que o simule ou uma plataforma *Linux*, *macOS* que contenha Mono (2017) instalado. Patrocinado pela *Microsoft*, *Mono* é uma implementação de código aberto do *Microsoft .NET Framework* baseado nos padrões *European Computer Manufacturers Association* (ECMA) para C# e *Common Language Runtime* (CLR). Tem como princípios ser uma plataforma de software projetada para permitir que os desenvolvedores criem facilmente aplicativos multiplataforma (*cross platform*).

Para se trabalhar com o código-fonte do *Mission Planner* também será necessária a instalação de uma IDE capaz de reconstruir o aplicativo feito na linguagem C#. Para este projeto, foi escolhida a IDE gratuita produzida pela *Microsoft* denominada *Visual Studio Community*.

3.2.1 DESCARREGANDO E INSTALANDO O MISSION PLANNER

Nesta sessão, serão vistas as etapas de como instalar todos os componentes necessários para colocar o ambiente do projeto em funcionamento. Inicia-se com os requerimentos do sistema:

- Windows Vista, 7, 8 ou 10
- Espaço suficiente em disco para poder executar a IDE Visual Studio Community (aproximadamente 7 GB)
- Visual Studio Community 2015

O primeiro passo é adquirir o executável do *Mission Planner*.

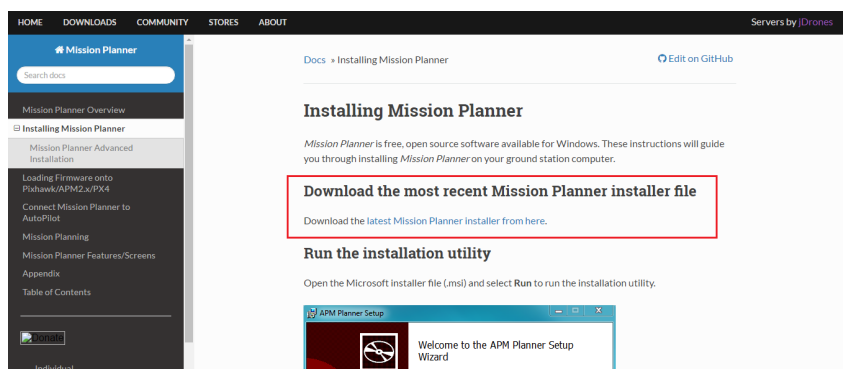


Figura 12 – Instalação do Mission Planner

Em seguida, é necessário instalar o IDE *Visual Studio 2015*.

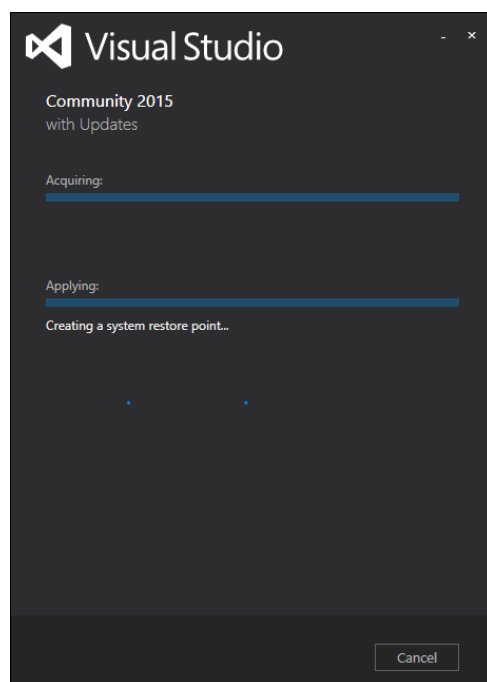


Figura 13 – Instalação do Visual Studio

Após descarregar o programa é preciso instalá-lo e reiniciar o sistema. Junto com a instalação do IDE também é necessário instalar o aplicativo *DirectX End-User Runtime*.

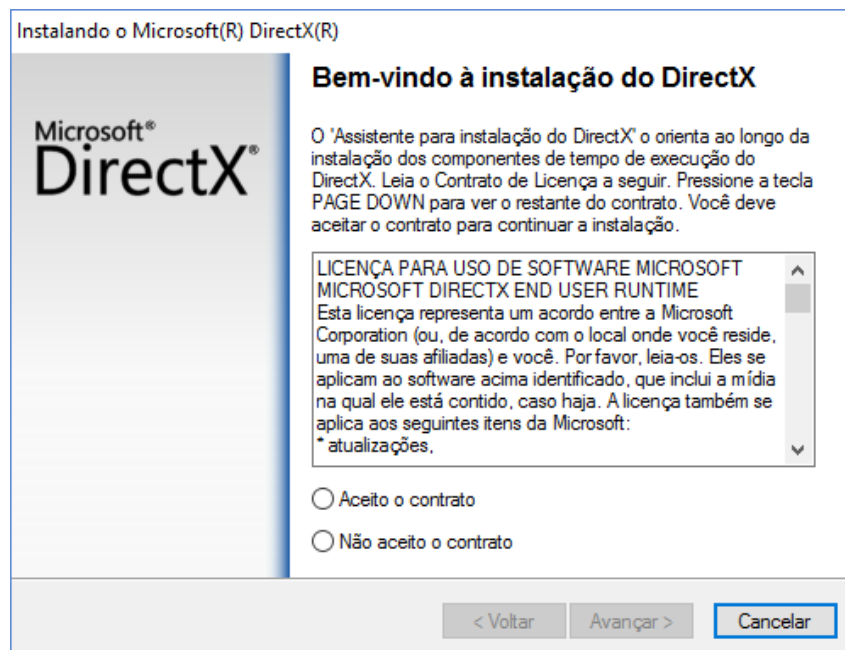


Figura 14 – Instalação do DirectX

3.2.2 TRABALHANDO COM O CÓDIGO-FONTE DO MISSION PLANNER

Após a instalação completa da IDE, é preciso adquirir o código-fonte do *Mission Planner* através do repositório armazenado no *GitHub*. A maneira mais simples e organizada de se realizar o *checkout* do repositório (armazená-lo no computador) é através do aplicativo *GitHub Desktop*. Para a utilização do mesmo, é necessária a criação de uma conta no site oficial do *GitHub*.

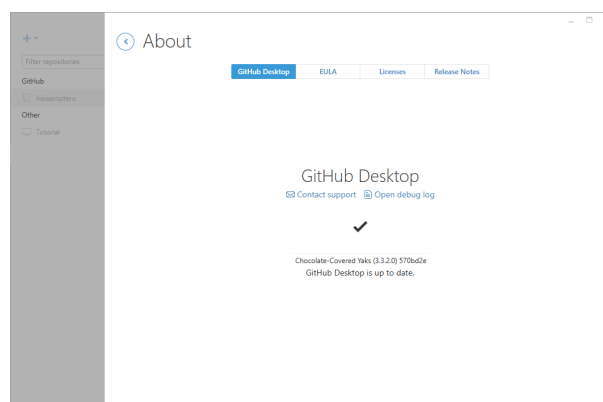


Figura 15 – GitHub Desktop

Depois de criar e entrar em uma conta do *GitHub* e após instalar o aplicativo, basta ir ao endereço do repositório do *Mission Planner* (MISSION PLANNER, 2017) e depois clicar no

botão verde *Clone or download* e em seguida *Open in Desktop*.

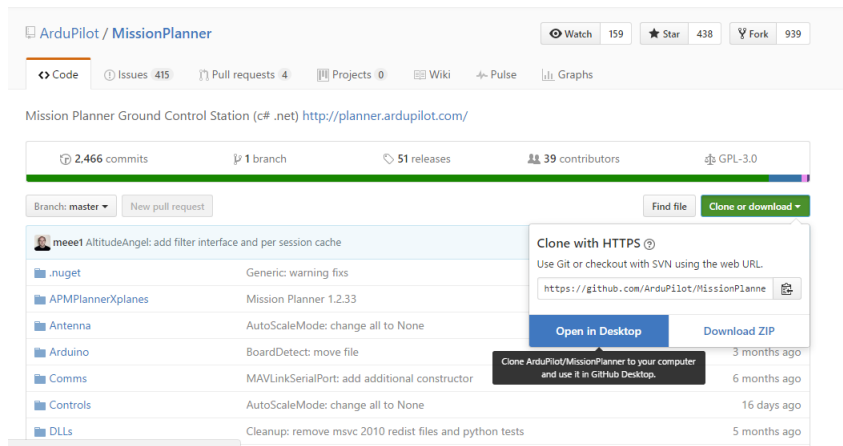


Figura 16 – Adquirindo repositório

Após clicar no botão, o aplicativo de *desktop* irá iniciar automaticamente e descarregar o repositório no diretório escolhido.

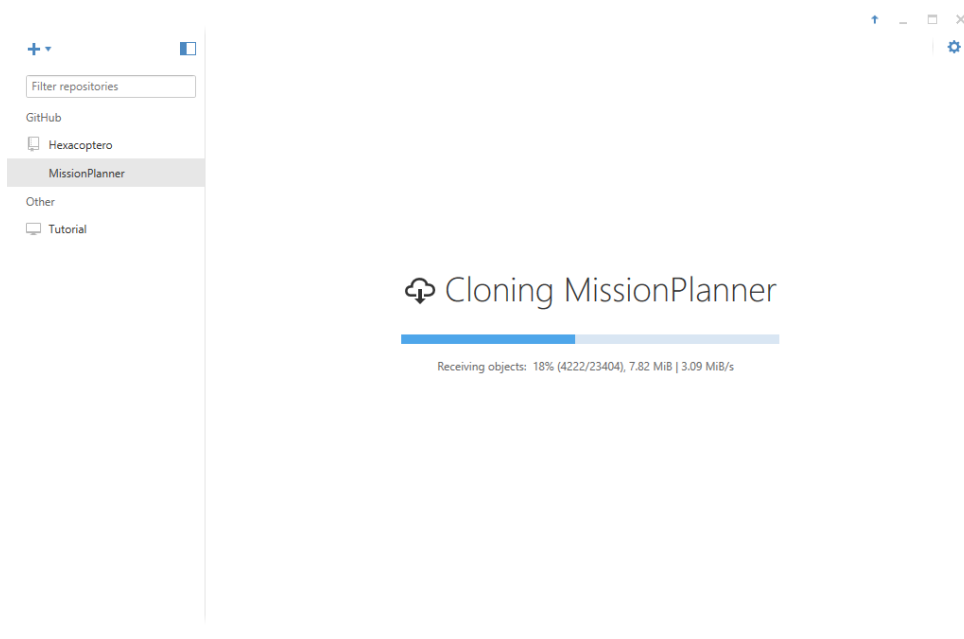


Figura 17 – Clonando repositório

Depois de descarregado, o repositório pode ser aberto diretamente do aplicativo do *GitHub* através do botão no canto superior direito escrito *Visual Studio*.

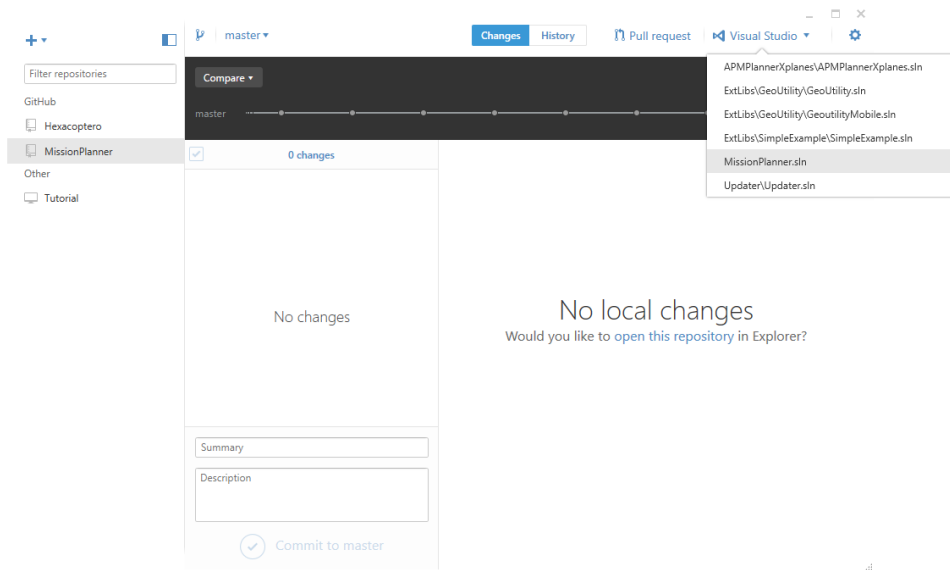


Figura 18 – Abrindo repositório diretamente do aplicativo GitHub Desktop

Após aberto o projeto, é necessário fazer pequenas alterações no mesmo para poder compilá-lo. Primeiramente precisa-se alterar o tipo de projeto de *Any CPU* para *x86* conforme mostrado abaixo.

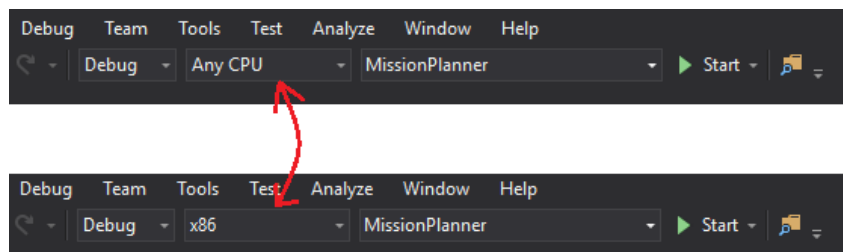


Figura 19 – Modificando o tipo de projeto

Em seguida, é necessário desabilitar a opção *Sign the ClickOnce manifests* que se encontra nas propriedades do projeto *Mission Planner* (botão direito, *Properties*), dentro da aba *Signing*.

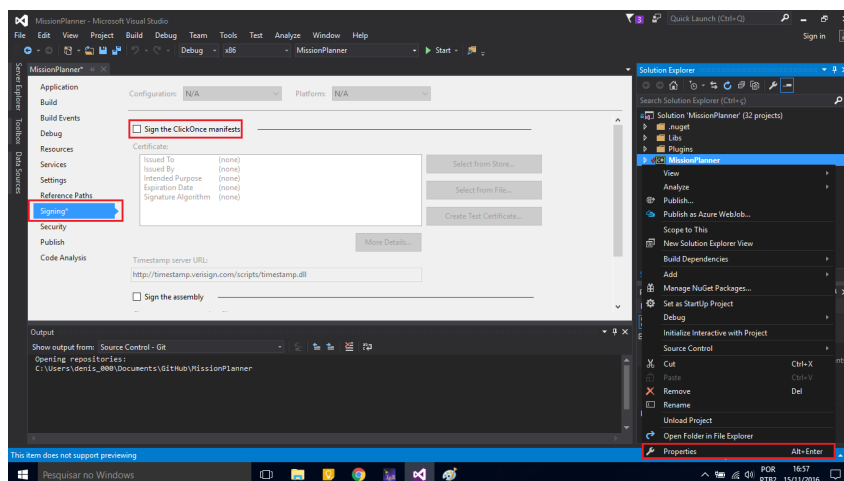


Figura 20 – Desabilitando Sign the ClickOnce manifests

Como existem bibliotecas que só veem acompanhadas no instalador do *Mission Planner*, é preciso referenciar a pasta onde o mesmo foi instalado para que o *Visual Studio* possa compilar o projeto. Para isso, é necessário clicar com o botão direito no projeto *Mission Planner*, depois em *Properties*, seguir para a aba *Reference Paths*, escrever o diretório onde foi instalado o programa e por último clicar em *Add Folder*.

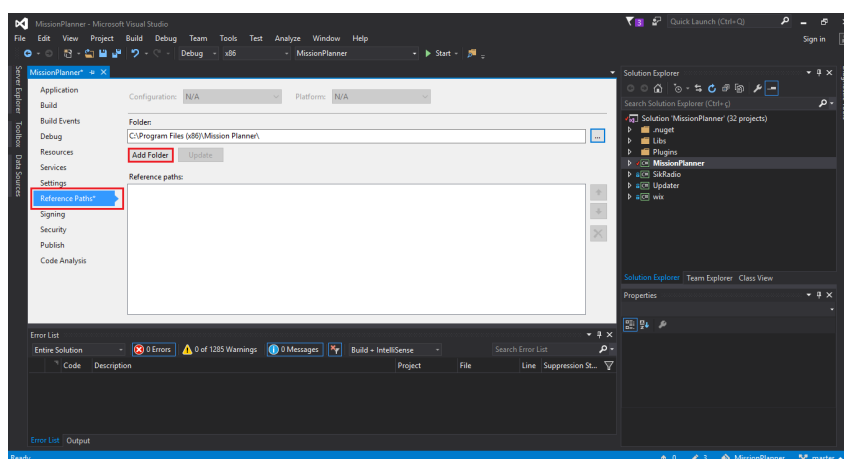


Figura 21 – Adicionando bibliotecas externas

Por fim, para compilar o projeto, é necessário ir ao menu *Build*, e depois clicar na opção *Batch Build*. Uma nova janela se abrirá como visto na Figura 23.

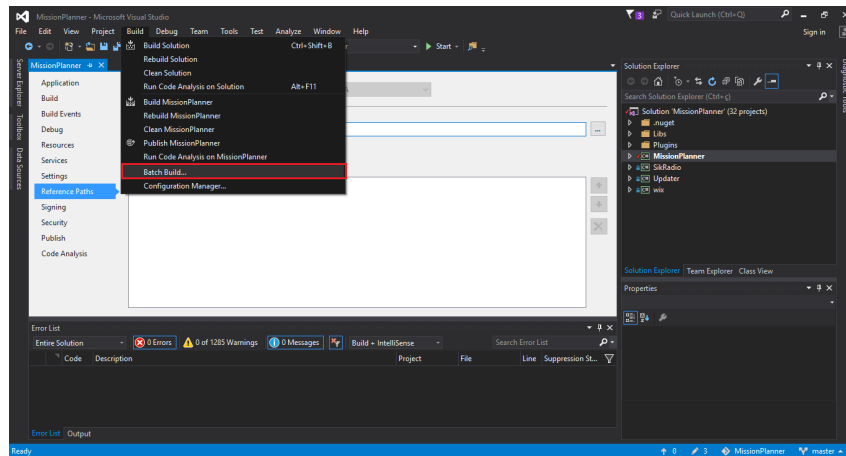


Figura 22 – Compilando projeto

Nesta nova janela é preciso clicar no botão *Select All* e depois *Rebuild*. Após alguns minutos o projeto estará pronto para ser executado. Para tal, basta clicar no botão com uma seta verde onde está escrito *Start*.

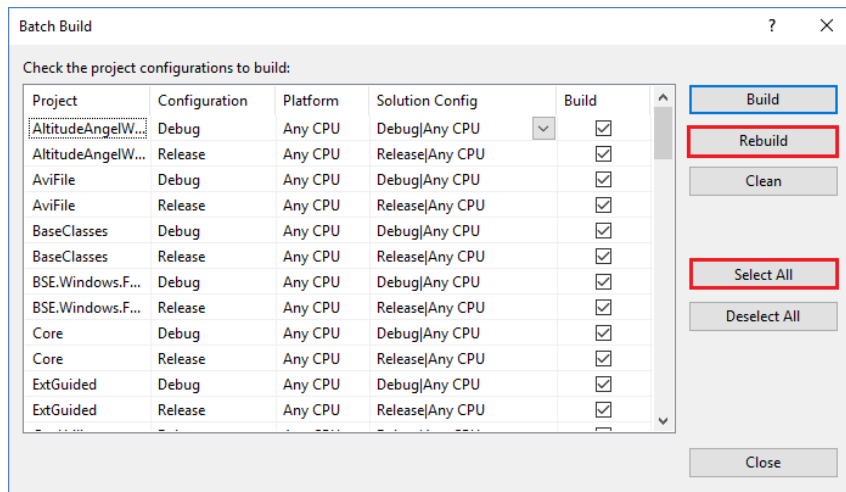


Figura 23 – Janela Batch Build

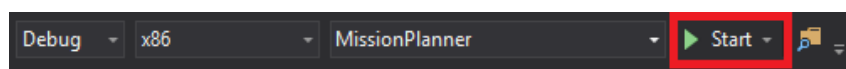


Figura 24 – Inicializando o projeto

3.3 MODIFICANDO O SOFTWARE MISSION PLANNER

Modificar o código do *Mission Planner* sem ter o perigo de causar algum dano ao código principal requer a implementação de um nova ramificação ao repositório (*branch*). *Branches* nada mais são do que pequenas referências que mantêm históricos independentes de *commits*. Ramificações são usadas normalmente para adicionar novas funcionalidades ao programa principal de forma isolada e segura.

3.3.1 ADICIONANDO UM NOVO BRANCH AO PROJETO DO MISSION PLANNER

Já com o *Visual Studio* aberto no projeto do *Mission Planner*, é necessário clicar na aba *Team Explorer* que fica na janela *Solution Explorer* no canto direito do programa. Nesta nova seção, deve-se clicar no botão *Branches*.

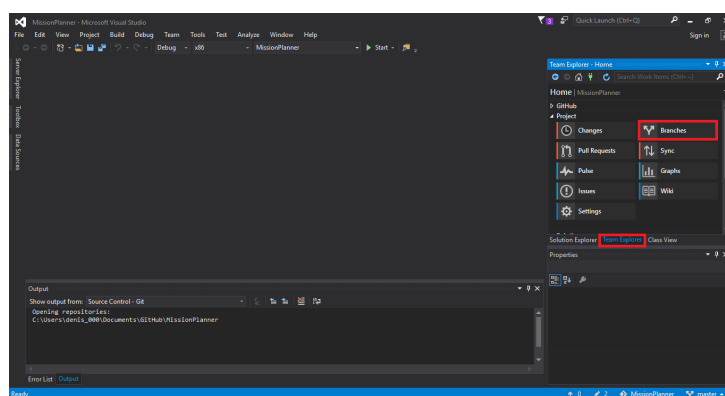


Figura 25 – a) Criando um novo branch

Agora clica-se com o botão direito sobre o repositório *Mission Planner* e depois em *New Local Branch From...*

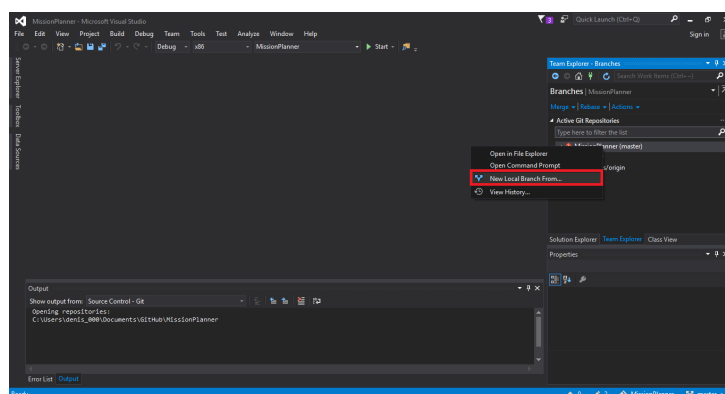


Figura 26 – b) Criando um novo branch

Logo em seguida, é preciso adicionar um nome para a nova ramificação, neste caso foi escolhido o nome “controlador_teclado” e escolher de qual *branch* este novo será formado. Como neste projeto só existe o *branch* principal (*master*) iremos usá-lo como referência. Após isto, basta clicar em *Create Branch*.

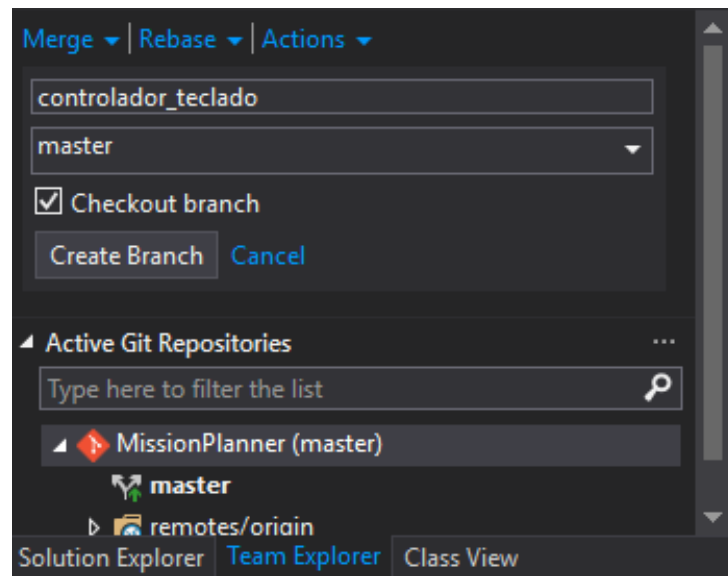


Figura 27 – c) Criando um novo branch

Com o novo *branch* criado, deve-se selecioná-lo para assim poder modificá-lo e posteriormente uni-lo ao *master* através de um *merge*. *Merge* é uma ação que tem como significado juntar todas as alterações feitas em um *branch* em outro.

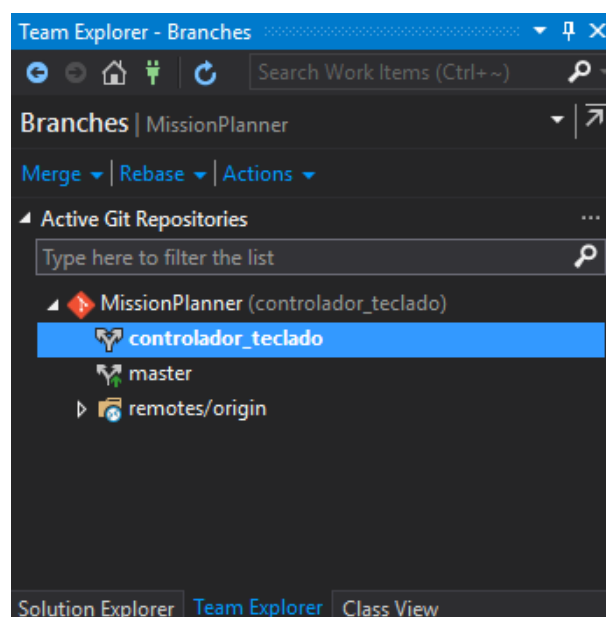


Figura 28 – Selecionando o novo branch

3.3.2 USANDO O NOVO BRANCH PARA MODIFICAR O CÓDIGO-FONTE

Primeiramente, antes de iniciar a implementação do novo método de pilotagem, foi estudada a melhor maneira de inseri-lo na interface do programa sem que o usuário final sentisse mudanças drásticas no fluxo de uso. A tela escolhida foi a mesma na qual já se encontra a opção de pilotagem através de um *joystick*. A mesma se encontra na aba CONFIG/TUNNING dentro das opções *Planner*.

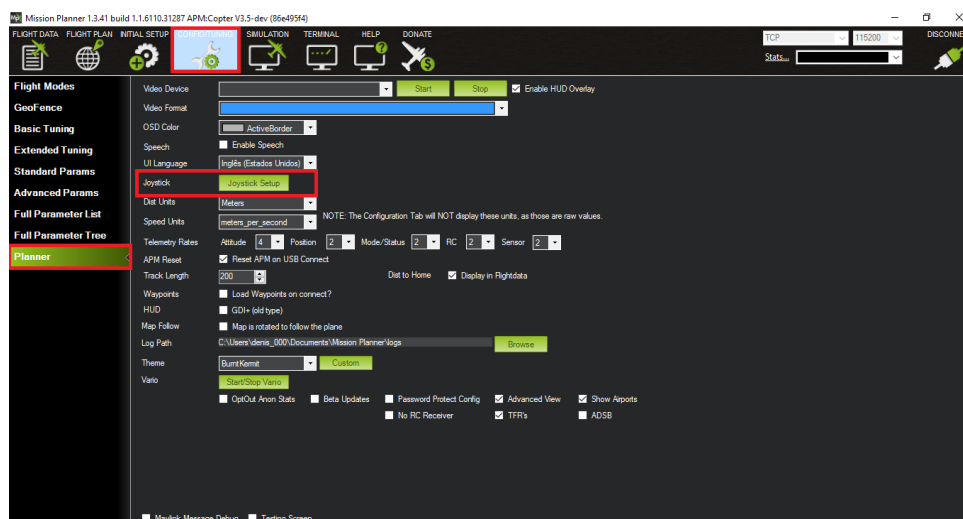


Figura 29 – Tela escolhida para inserir o novo método

A tela modificada irá acrescentar apenas um novo botão denominado *Keyboard Setup*, seguindo o modelo já existente.



Figura 30 – Modelo da nova tela

3.3.2.1 Inserindo o botão de configurações do teclado

A tela a ser alterada se encontra dentro do diretório *GCSViews/ConfigurationView* e sua classe é denominada no código-fonte por *ConfigPlanner.cs*.

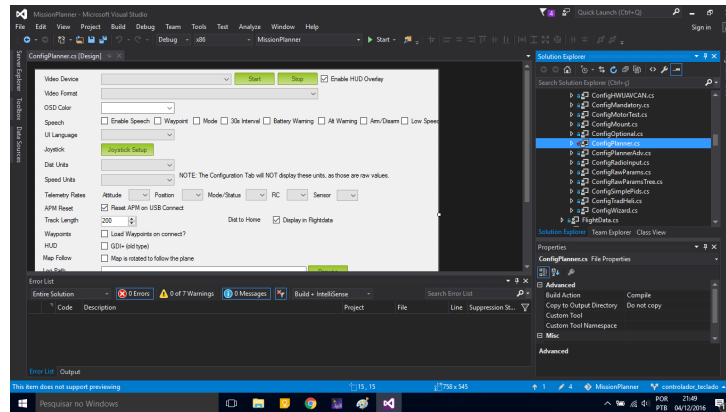


Figura 31 – Tela Planner vista do código-fonte

Para se adicionar a nova fileira que conterá o novo botão, foi necessário aumentar a altura da tela para 545 pixels usando a propriedade *Height*, presente dentro de *Size* na janela *Properties* localizada no canto direito inferior do *Visual Studio*.

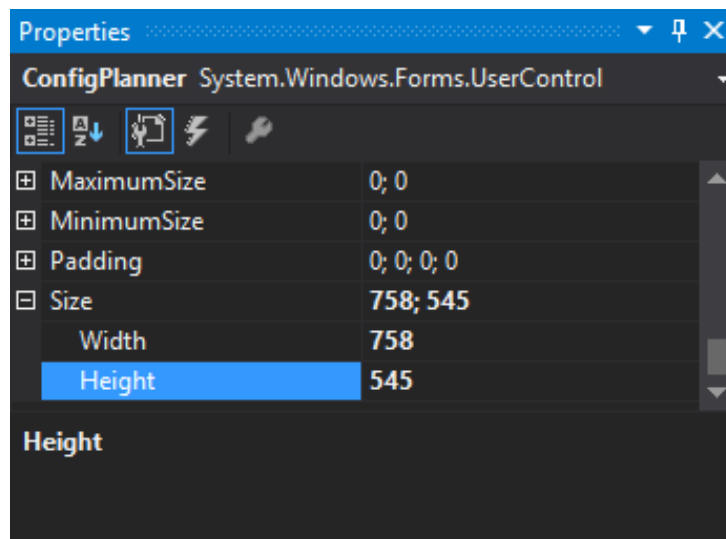
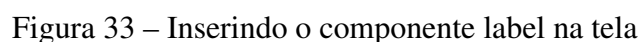


Figura 32 – Aumentando altura da tela

Após aumentar o tamanho da tela, foi preciso arrastar todos os componentes localizados abaixo do botão *Joystick Setup* para liberar espaço ao novo botão e sua respectiva etiqueta (*label*). Para inserir uma nova *label*, clica-se na aba *Toolbox* que se encontra a esquerda da janela principal da Ambiente de Desenvolvimento Integrado (IDE em inglês), pesquisamos na caixa de seleção pelo componente desejado e o selecionamos.



The screenshot shows the Visual Studio Toolbox on the left side of the interface. The 'MissionPlanner.Controls Components' group is expanded and highlighted with a red rectangle. Below this group, the 'MyButton' control is highlighted with a blue background. The main workspace on the right shows a 3D model of a quadcopter in a flight simulator environment.

Figura 34 – Inserindo o botão na tela

3.3.2.2 Inserindo a interface de configurações do teclado

Para continuar, é preciso adicionar um evento ao novo botão para que o mesmo seja capaz de iniciar uma nova janela de configuração de controle via teclado ao clique do usuário. Para tal, navega-se até a janela de propriedades localizada no canto inferior direito, e clica-se no ícone em formato de raio que é o gerenciador de eventos do componente. Ao clicar no gerenciador, a janela mostrará todos os eventos possíveis de serem adicionados ao componente. Neste momento, o foco é o evento *Click* que é acionado toda vez que o usuário clicar no botão. Seguindo os padrões de nomenclatura do projeto, adicionamos o novo evento *BUT_Keyboard_Click*.

Depois de preparado o código para a inserção do novo evento é preciso criar as classes que serão necessárias para a implementação do controle via teclado. Inicia-se essa nova etapa criando-se a classe *KeyboardSetup* que será responsável pela aparência da janela de configuração.

Para a implementação desta nova classe, primeiro é adicionada uma pasta ao projeto denominada *Keyboard* clicando com o botão direito na solução (MissionPlanner.sln) e em seguida em *Add -> New Folder*. Após isso, a nova classe é inserida dentro desta pasta clicando na mesma e depois em *Add -> Windows Form*.

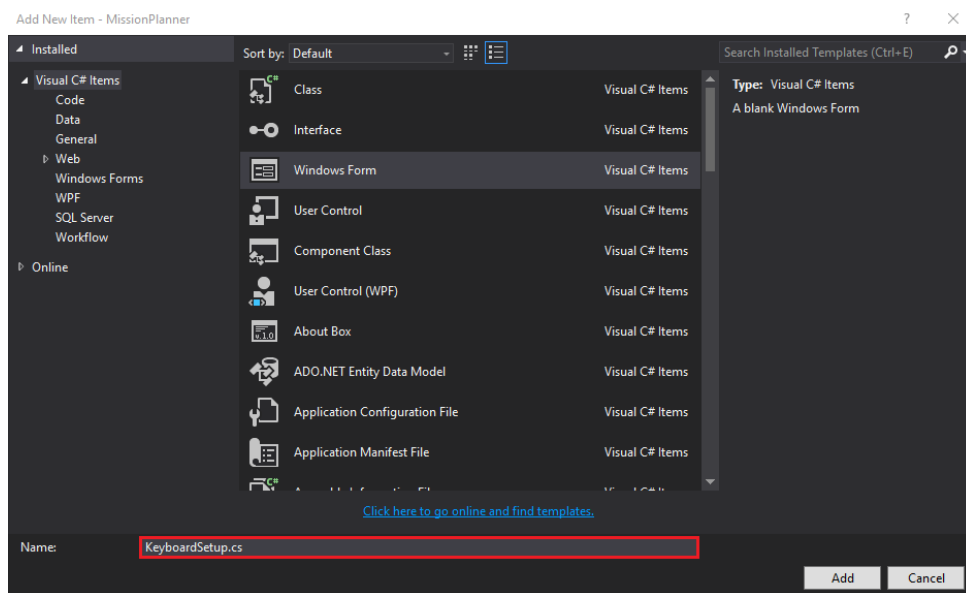


Figura 35 – Inserindo nova classe ao projeto

Após criada a nova interface, é necessário que o evento do botão inicialize a mesma. Para tal, o código abaixo é inserido ao método *BUT_Keyboard_Click*.

```
private void BUT_Keyboard_Click(object sender, EventArgs e)
{
    Form keyboard = new KeyboardSetup();
    ThemeManager.ApplyThemeTo(keyboard);
    keyboard.Show();
}
```

}

Código 3.1 – Método BUT_Keyboard_Click

Tendo como base a interface utilizada para as configurações de joystick, é feita a interface para o teclado. A classe completa pode ser vista no Anexo A.

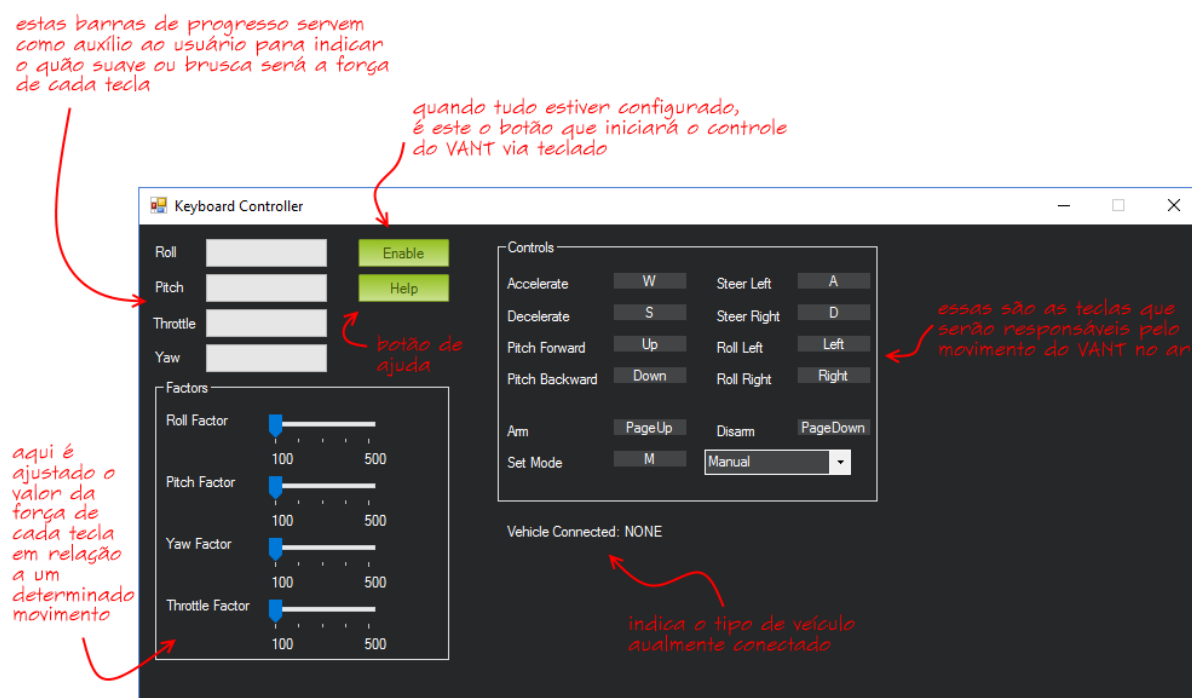


Figura 36 – Tela de configurações do teclado

3.3.2.3 Criando a classe Keyboard

Feita a interface, o próximo passo é armazenar todas as configurações escolhidas pelo usuário em uma classe única para que essas informações não sejam perdidas quando a janela de configurações for fechada. Para isso, é criada a classe *Keyboard*. Esta irá ser responsável por armazenar os dados do usuário como também por realizar uma resposta ao pressionamento de uma tecla. Para conter os dados, a classe necessita ser construída de modo a possuir todas as variáveis que serão utilizadas pelo usuário. Abaixo encontra-se a lista destas variáveis.

- Teclado habilitado/desabilitado
- Acelerar
- Desacelerar
- Yaw a esquerda
- Yaw a direita
- Potência Yaw

- Roll a esquerda
- Roll a direita
- Potência Roll
- Pitch para frente
- Pitch para trás
- Potência Pitch
- Armar
- Desarmar
- Mudar o modo de voo
- Tipo de modo de voo
- Potência dos motores

Estruturada a classe para armazenar estes dados, foi implementado na interface *KeyboardSetup* um evento, denominado *BUT_Enable_Click*, no clique do botão de habilitar (*Enable*) para que os dados da interface sejam guardados na classe *Keyboard* (Anexo B).

Após armazenados os dados, é preciso agora implementar métodos para detectar todos os pressionamentos de tecla feitos pelo usuário tanto dentro do aplicativo *Mission Planner* como fora. Esta etapa foi um desafio, pois foi necessário descobrir alguma maneira de conseguir detectar o pressionar de uma tecla fora do ambiente do aplicativo. Isto é crucial devido ao fato de o usuário querer pilotar o VANT com outras janelas abertas como *logs*, gráficos ou para obtenção de dados. Após algum tempo dedicado a pesquisas, foi possível encontrar uma classe com todos os requisitos fundamentais na plataforma online de código aberto *GitHub* no repositório do usuário Afkbio (2012).

Através da implementação do arquivo ao projeto e sua conexão a classe *Keyboard* foi possível realizar testes para conferir se as teclas eram realmente lidas de qualquer parte do sistema.

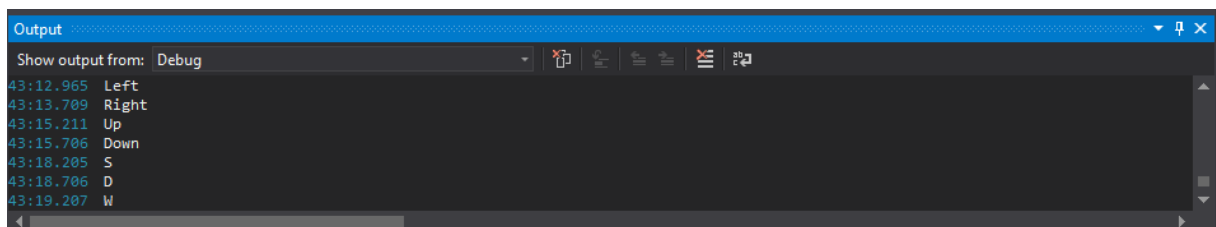


Figura 37 – Tela de debug da IDE com as teclas pressionadas

3.3.2.4 Criando comunicação entre teclado e VANT

Agora com a interface capturando teclas de maneira global, foi preciso, como última etapa deste processo de implementação, descobrir como enviar estes dados do teclado para ao

VANT conectado ao *Mission Planner*.

Espelhando-se no modo como foi implementado os comandos de envio do Joystick para o VANT, foram utilizados os mesmo princípios para o envio dos comandos do teclado.

A ideia baseia-se na criação de uma *thread* na classe principal do sistema, denominada *MainV2*, no momento da inicialização da aplicação. Este algoritmo verifica continuamente se o teclado está habilitado. Caso esteja, o código verifica se há algum veículo conectado a aplicação e envia os dados sobrescritos pelo teclado aos respectivos canais do drone.

```
if (MainV2.comPort.MAV.cs.rcoverridech1 != 0)
rc.chan1_raw = MainV2.comPort.MAV.cs.rcoverridech1;
if (MainV2.comPort.MAV.cs.rcoverridech2 != 0)
rc.chan2_raw = MainV2.comPort.MAV.cs.rcoverridech2;
if (MainV2.comPort.MAV.cs.rcoverridech3 != 0)
rc.chan3_raw = MainV2.comPort.MAV.cs.rcoverridech3;
if (MainV2.comPort.MAV.cs.rcoverridech4 != 0)
rc.chan4_raw = MainV2.comPort.MAV.cs.rcoverridech4;
if (MainV2.comPort.MAV.cs.rcoverridech5 != 0)
rc.chan5_raw = MainV2.comPort.MAV.cs.rcoverridech5;
if (MainV2.comPort.MAV.cs.rcoverridech6 != 0)
rc.chan6_raw = MainV2.comPort.MAV.cs.rcoverridech6;
if (MainV2.comPort.MAV.cs.rcoverridech7 != 0)
rc.chan7_raw = MainV2.comPort.MAV.cs.rcoverridech7;
if (MainV2.comPort.MAV.cs.rcoverridech8 != 0)
rc.chan8_raw = MainV2.comPort.MAV.cs.rcoverridech8;
```

Código 3.2 – Parte da implementação da thread do teclado

Como visto no código acima, são oito os números de canais de rádio a serem sobrescritos. Cada rádio é responsável por um tipo de movimento do VANT. Por curiosidade, foi visto que os valores repassados aos canais da aeronave são em microssegundos, portanto, repassando um valor de mil significa que a aeronave ativará o respectivo canal por mil microssegundos. Quando o valor repassado é zero, isto indica ao VANT que o canal não está mais sendo sobrescrito por algum dispositivo e portanto o drone deve redirecionar a sua escuta ao controlador de rádio comum. O envio do valor zero é feito toda vez que o usuário desabilita o controle do veículo via teclado. O código completo da *thread* pode ser visto no Anexo C.

3.3.3 TESTANDO A NOVA IMPLEMENTAÇÃO DO SOFTWARE

Com as mudanças devidamente implementadas, foram feitos testes dessa nova funcionalidade utilizando um simulador SITL que existe dentro da própria aplicação do *Mission Planner*.

Para iniciar uma sessão de simulação pelo *Mission Planner*, basta clicar na aba *Simulation* e depois escolher o tipo de veículo que se deseja simular. No caso deste projeto, usamos o *Multirotor*. Após o simulador descarregar todos os arquivos necessários e conectar-se a aplicação. Através do manual do *ArduPilot*, verificou-se que era necessário ajustar o valor de um parâmetro para que o drone virtual pudesse armar-se. O parâmetro ajustado foi o *FRAME_CLASS* e o valor foi trocado de zero para um que significa um drone de quatro motores (ARDUPILOT, 2016c). Na imagem abaixo é possível ver uma simulação feita na aplicação. O veículo se encontra sobre o oceano Atlântico próximo a África devido a falta de um GPS no sistema virtual.

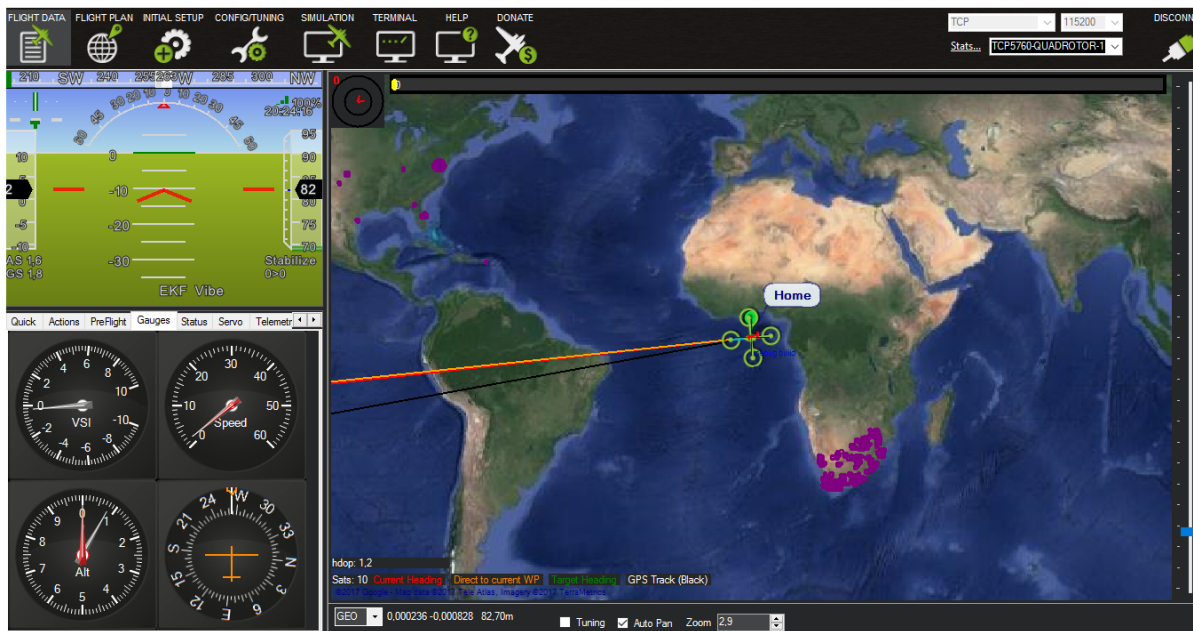


Figura 38 – Mission Planner rodando uma simulação de multirrotor

3.4 APRIMORANDO OS DISPOSITIVOS DE LOCALIZAÇÃO E COMUNICAÇÃO

Depois de finalizada a parte de software, foram necessários alguns ajustes em relação à parte de hardware. Nos primeiros testes feitos com o VANT, percebeu-se que o mesmo tinha grandes dificuldades, mesmos em ambientes a céu aberto, em se localizar espacialmente e também globalmente através do compasso eletrônico incluso no chip do *ArduPilot* devido ao ruído provocado pelo campo eletromagnético produzido pelas altas correntes (cerca de 25A) que passam pelos cabos de energia e bateria localizados logo abaixo da placa. Para resolver o problema, teve-se que adicionar um novo componente de geolocalização (GPS) que também continha um compasso eletrônico.

Feita a alteração, pode-se testar o drone pilotando-o em um campo de futebol via rádio-controle. Após alguns voos curtos, pode-se notar outro grave problema. O sinal de comunicação do rádiomodem da telemetria se perdia a aproximadamente 20 metros de distância. Medida extremamente restrita quando se falando de veículos voadores.

Demorou um tempo até se descobrir que o problema realmente se tratava da antena conectada no drone, devido à frequência estar errada: a antena que veio acompanhando o rádiomodem era de 5,2GHz, enquanto a frequência do dispositivo era de 433MHz. Após algumas pesquisas, foi possível montar manualmente uma antena que foi acoplada no lugar da antiga.

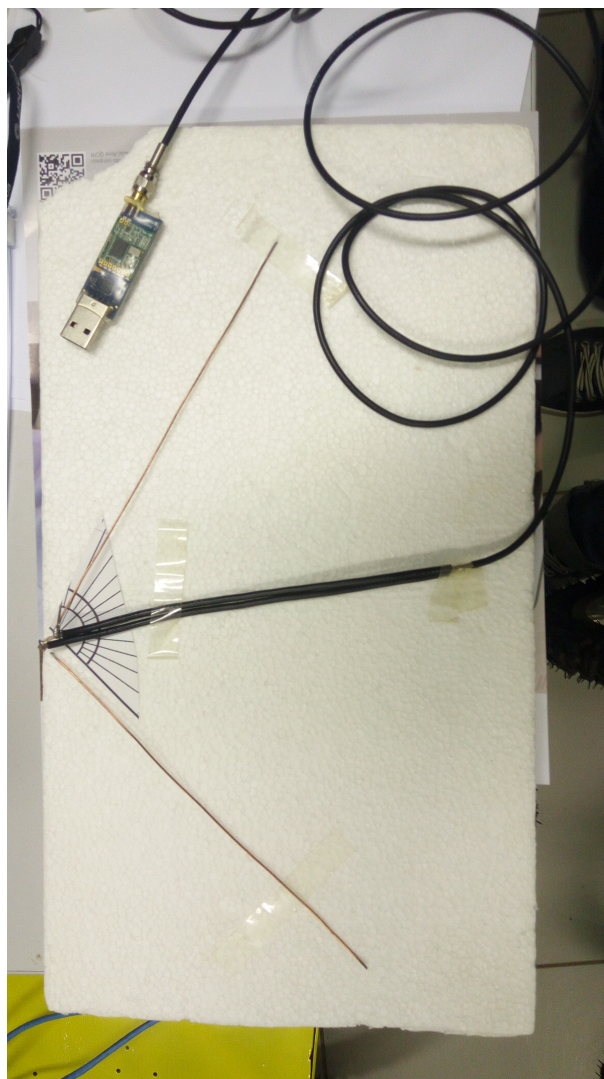


Figura 39 – Antena construída manualmente

Esta foi capaz de se comunicar com a estação em solo (um notebook) de uma distância de até 160 metros. Ainda longe do ideal, foi construída uma segunda antena que foi acoplada agora a estação em solo. Testou-se novamente e o resultado obtido foi que com a mesma distância percorrida anteriormente, a perda de sinal permanecia em apenas 10%.

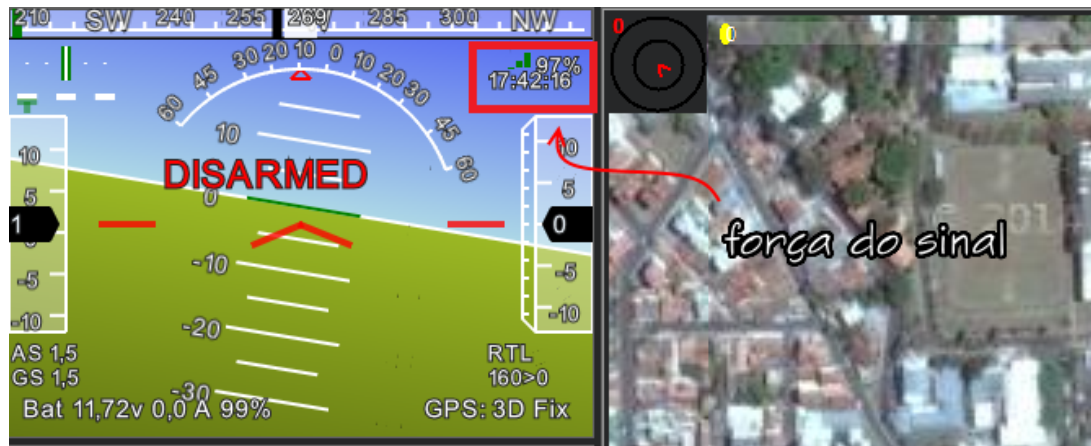


Figura 40 – Perda de sinal com as novas antenas em uma distância de aproximadamente 160m

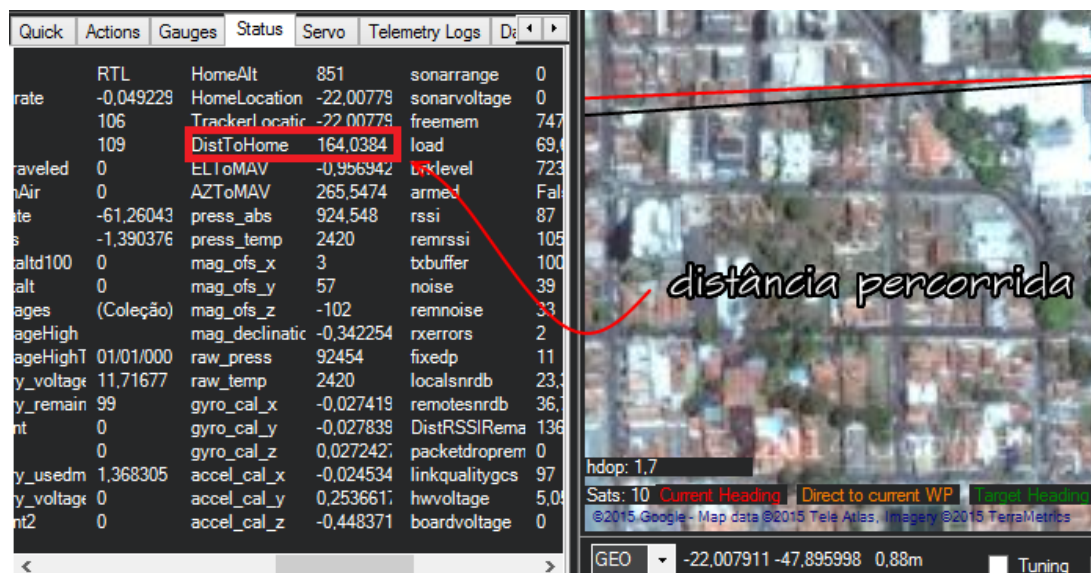


Figura 41 – Distância percorrida

3.5 TESTES DE CONFIABILIDADE

3.5.1 TESTES COM AS NOVAS ANTENAS

Depois de ter as antenas aprimoradas, o VANT foi testado primeiramente com um joystick conectado a estação em solo utilizando o *Mission Planner*. Inicialmente foi necessário descobrir a sequência certa de comandos para armar a aeronave para poder controlá-la, no qual se trata em deixar o acelerador (*throttle*) no mínimo e o leme para a direita (yaw no máximo) por cinco segundos (ARDUPILOT, 2016a). Após vários testes com o joystick, notou-se que as antenas estavam funcionando corretamente.

3.5.2 TESTES DE FAILSAFE

Após os testes de confiabilidade, o VANT foi amarrado a uma mesa para que se pudesse realizar os testes de *failsafe*. Estes são eventos que ocorrem quando o controle do veículo é perdido. Existem vários tipos de *failsafe* (ARDUPILOT, 2016b), neste caso será discutido o evento do tipo *Radio Failsafe* (ARDUPILOT, 2016d), no qual o contato entre o rádiocontrole do piloto e o receptor do controlador de voo é perdido. Se habilitado e configurado corretamente, este evento pode ser disparado nos seguintes casos:

- O piloto desliga o rádiocontrole
- O veículo ultrapassa o alcance do rádiocontrole (geralmente em torno de 500m à 700m)
- O receptor perde energia
- Os fios que ligam o receptor ao controlador de voo estão quebrados

Quando um *Radio Failsafe* é disparado, o VANT pode ter as seguintes reações:

- Não fazer nada se o veículo já estiver desarmado
- Desarmar os motores imediatamente se o veículo estiver aterrado, em modo estabilizado ou de acrobacia e o acelerador do piloto automático estiver em zero
- Retorna a posição inicial de lançamento (*Return-to-Launch* ou RTL) se o veículo dispuser de uma trava de GPS e estiver a mais de dois metros da posição inicial
- Continuar com a missão se o veículo estiver no modo automático (*AUTO*) e o parâmetro “FS_THR_ENABLE” estiver configurado como “Enabled Continue with Mission in Auto Mode”.
- Aterrissar o veículo se:
 - Não tiver uma trava GPS
 - Estiver a dois metros de distância da posição inicial
 - O parâmetro “FS_THR_ENABLE” estiver configurado como “Enabled Always Land”

Se o *failsafe* for cancelado, isto é, o transmissor e o receptor recuperarem o contato, o VANT permanecerá em seu modo de voo atual. O mesmo não retornará automaticamente ao modo de voo que estava ativo antes do *failsafe* ser disparado. Isto significa que se, por exemplo, o veículo estivesse em *Loiter* quando ocorreu o *failsafe* e o modo de voo foi automaticamente alterado para RTL, mesmo depois do receptor recuperar o contato, o veículo permanecerá em RTL. Se o piloto desejar retomar o controle em modo *Loiter*, o mesmo precisará mudar seu interruptor de modo de voo para outra posição e então de volta para *Loiter*.

Os testes a seguir foram feitos conforme as orientações sugeridas (ARDUPILOT, 2016f). Primeiramente foi feito o teste desabilitando a comunicação do teclado no meio de um voo com o rádiocontrole desligado seguindo os passos:

- Desligar o rádiocontrole
- Abrir o *Mission Planner*, abrir as configurações do teclado (*Keyboard Setup*), habilitar o modo de controle por teclado e verificar que o mesmo está sobrescrevendo os sinais de transmissão via rádio.
- Armar o veículo e mudar para o modo *Stabilize* ou *Loiter* e aumentar o acelerador (*throttle*)
- Desabilitar o controle por teclado
- “Failsafe” deverá aparecer na interface e o veículo deve mudar para o modo *LAND* ou *RTL*

O segundo teste também foi feito desabilitando o controle do teclado, porém agora com o rádiocontrole ligado:

- Ligar o rádiocontrole e verificar que o mesmo pode controlar o veículo normalmente
- Abrir o *Mission Planner*, abrir as configurações do teclado (*Keyboard Setup*), habilitar o modo de controle por teclado e verificar que o mesmo está sobrescrevendo os sinais do rádiocontrole.
- Armar o veículo e mudar para o modo *Stabilize* ou *Loiter* e aumentar o acelerador (*throttle*)
- Desabilitar o controle por teclado
- O veículo deve permanecer em seu modo atual (*Stabilize* ou *Loiter*), mas os controles devem retornar ao rádiocontrole

3.5.3 TESTE FINAL EM CAMPO ABERTO

Após verificados os resultados nos testes de *failsafe*, confirmando que caso acontecesse qualquer problema entre a comunicação da estação em solo e o VANT, o controle deste seria retornado ao rádiocontrole automaticamente, foi realizado o teste final pilotando o drone através do teclado do notebook em um campo de futebol. O teste foi realizado em um dia ensolarado e sem vento. Durante o experimento, notou-se nenhuma discrepância em comparação a pilotagem com o joystick em relação ao tempo de resposta da aeronave. No entanto, nos momentos finais do teste durante a aterrissagem do VANT, ocorreu um disparo do anti-vírus do notebook informando que uma varredura havia sido completada. Isto fez com que o controle do teclado fosse perdido momentaneamente fazendo com que a aeronave capotasse e ficasse de ponta-cabeça no solo.

4 CONSIDERAÇÕES FINAIS

4.1 CONCLUSÃO

Este trabalho consistiu na pesquisa acerca da possibilidade de se pilotar um VANT através da comunicação sem fio em tempo real entre uma estação em solo e um controlador de voo, abrangendo as etapas de revisão bibliográfica, desenvolvimento do software e aprimoramento da interface de comunicação para a obtenção do resultado.

A revisão bibliográfica foi apresentada de forma a explicar os conceitos envolvidos e os softwares utilizados, destacando a importância nos avanços do setor de aeronaves não-tripuladas, que permitiram uma redução no custo de construção e desenvolvimentos destes veículos.

As classes implementadas no aplicativo do *Mission Planner* via *Visual Studio* contemplam uma estrutura capaz de captar teclas pressionadas por um usuário de qualquer parte do sistema operacional em andamento, convertendo esta tecla para um valor capaz de ser entendido por um VANT e enviar estes dados para o veículo através de uma *thread* contínua.

Para o desenvolvimento completo do trabalho, não foram o bastante os conhecimentos inicialmente previstos, foi necessária muita pesquisa, leitura de documentos e auxílio do orientador para resolução de problemas dos componentes utilizados pelo VANT, tal como o refinamento das antenas utilizadas.

Apesar dos problemas iniciais relacionados à geolocalização e a fragilidade do sistema de comunicação em perder seu sinal com a outra ponta com relativa facilidade, o projeto demonstrou-se bem sucedido a partir do momento em que estas adversidades foram possíveis de ser resolvidas.

Após a realização dessas considerações, pode-se afirmar que o projeto cumpriu o objetivo proposto em desenvolver um novo sistema de controle para VANTs diretamente pelo software que executa a estação em solo, por meio de um modem sem fio e um teclado.

4.2 CONTRIBUIÇÕES

O processo de desenvolvimento do projeto abordou áreas tal como a utilização de um sistema de controle de versões utilizando *git* através do *software de desktop* do *GitHub*, servindo de material de apoio e solução para quem possa vir a trabalhar com este tipo de sistema. Além disso, as alterações feitas no código-fonte do *Mission Planner* e as etapas necessárias para realizá-las podem contribuir para futuras aplicações neste mesmo software ou em outros que utilizem o protocolo *MAVLink*. As informações contidas neste trabalho devem servir de ponto de

partida para que em trabalhos futuros uma versão mais robusta da comunicação entre estação em solo e VANT possa ser implementada e então testadas com mais segurança no veículo em movimento.

4.3 TRABALHOS FUTUROS

Deixa-se como sugestão para pesquisas futuras na área o estudo da possibilidade de pilotar-se um VANT utilizando-se um dispositivo móvel tal como um *smartphone*. Sabe-se ser possível conectar em alguns modelos de celulares *Android* o modem de telemetria utilizado neste projeto. Além disso, já existem aplicativos disponíveis que possuem funcionalidades semelhantes ao *Mission Planner*, tal como o Tower (2016).



Figura 42 – Modem de telemetria utilizado no projeto conectado a um smartphone

REFERÊNCIAS

- AFKBIO. *Repositório com código de detecção global de pressionamento de teclas*. [S.l.], 2012. Disponível em: <<https://github.com/Afkbio/diabloitemcapture/blob/master/globalKeyboardHook.cs>>. Acesso em: 20 mai. 2017.
- APM 2.5 and 2.6 Overview. 2016. Disponível em: <<http://ardupilot.org/copter/docs/common-apm25-and-26-overview.html#common-apm25-and-26-overview>>. Acesso em: 13 mai. 2017.
- APM PLANNER. *APM Planner Home*. [S.l.], 2016. Disponível em: <<http://ardupilot.org/planner2/>>. Acesso em: 13 mai. 2017.
- ARDUPILOT. *Arming the motors*. 2016. Disponível em: <http://ardupilot.org/copter/docs/arming_the_motors.html>. Acesso em: 20 mai. 2017.
- ARDUPILOT. *Failsafe*. 2016. Disponível em: <<http://ardupilot.org/copter/docs/failsafe-landing-page.html>>. Acesso em: 20 mai. 2017.
- ARDUPILOT. *Parâmetro FRAME_CLASS*. 2016. Disponível em: <<http://ardupilot.org/copter/docs/parameters.html>>. Acesso em: 20 mai. 2017.
- ARDUPILOT. *Radio Failsafe*. 2016. Disponível em: <<http://ardupilot.org/copter/docs/radio-failsafe.html>>. Acesso em: 20 mai. 2017.
- ARDUPILOT. *Supported AutoPilot Controller Boards*. 2016. Disponível em: <<http://ardupilot.org/dev/docs/supported-autopilot-controller-boards.html#supported-autopilot-controller-boards>>. Acesso em: 15 mai. 2017.
- ARDUPILOT. *Testing the failsafes*. 2016. Disponível em: <<http://ardupilot.org/copter/docs/flying-with-a-joystickgamepad-instead-of-rc-controller.html#testing-the-failsafes>>. Acesso em: 20 mai. 2017.
- ÅSTRÖM, K. J.; HÄGGLUND, T. *PID controllers: theory, design, and tuning*. [S.l.]: ISA Research Triangle Park, NC, 1995.
- BODDINGTON, D. *Radio-Controlled Model Aircraft*. [S.l.]: Crowood Press, 2004.
- CARDOSO, A. et al. Técnicas de engenharia de software e visualização da informação de dados capturados por aeronaves remotamente pilotadas associadas na pecuária de precisão. *N/A*, 2016.
- CASBEER, D. W. et al. Cooperative forest fire surveillance using a team of small unmanned air vehicles. *International Journal of Systems Science*, v. 37, n. 6, p. 351–360, 2006.
- CAVOUKIAN, A. Privacy and drones: Unmanned aerial vehicles. *Research Group of The Office Privacy Commission of Canada*, 2012. Disponível em: <<https://www.ipc.on.ca/wp-content/uploads/Resources/pbd-drones.pdf>>.
- DEMOLINARI, H. C. *Projeto de construção um drone hexacóptero*. Dissertação (Mestrado) — Universidade Federal Fluminense, 2016.

DOHERTY, P.; RUDOL, P. A uav search and rescue scenario with human body detection and geolocalization. In: *AI 2007: Advances in Artificial Intelligence*. [S.l.]: Springer, 2007. p. 1–13.

FEDERAL AVIATION ADMINISTRATION. *Advanced Avionics Handbook*. [S.l.], 2009. Disponível em: <https://www.faa.gov/regulations_policies/handbooks_manuals/aviation/advanced_avionics_handbook/media/aah_ch04.pdf>.

GIRARD, A. R.; HOWELL, A. S.; HEDRICK, J. K. Border patrol and surveillance missions using multiple unmanned air vehicles. In: *IEEE. Decision and Control, 2004. CDC. 43rd IEEE Conference on*. [S.l.], 2004. v. 1, p. 620–625.

GIT. *A Short History of Git*. [S.l.], 2005. Disponível em: <<https://git-scm.com/book/en/v2/Getting-Started-A-Short-History-of-Git>>.

GIT. *Git Home*. [S.l.], 2017. Disponível em: <<https://git-scm.com/>>.

GIT. *Gravando Alterações no Repositório*. [S.l.], 2017. Disponível em: <<https://git-scm.com/book/pt-br/v1/Git-Essencial-Gravando-Altera%C3%A7%C3%B5es-no-Reposit%C3%B3rio>>. Acesso em: 17 mai. 2017.

GITHUB. *GitHub Home*. [S.l.], 2017. Disponível em: <<https://github.com/>>. Acesso em: 18 mai. 2017.

GITHUB. *Hello World*. [S.l.], 2017. Disponível em: <<https://guides.github.com/activities/hello-world/>>. Acesso em: 18 mai. 2017.

GONÇALVES, J. E. et al. Veículo aéreo não tripulado para perfilamento atmosférico em alta resolução. *N/A*, 2006.

LAURENT, A. M. S. *Understanding Open Source and Free Software Licensing: Guide to Navigating Licensing Issues in Existing & New Software*. [S.l.]: O'Reilly Media, Inc, 2004.

MAVLINK. *Repositório MAVLink*. 2017. Disponível em: <<https://github.com/mavlink/mavlink>>. Acesso em: 21 mai. 2017.

MEIER, L. *MAVLink*. 2009. Disponível em: <<http://qgroundcontrol.org/mavlink/start>>. Acesso em: 21 mai. 2017.

MISSION PLANNER. *Mission Planner Home*. [S.l.], 2016. Disponível em: <<http://ardupilot.org/planner/>>. Acesso em: 13 mai. 2017.

MISSION PLANNER. *Repositório Mission Planner*. [S.l.], 2017. Disponível em: <<https://github.com/ArduPilot/MissionPlanner>>. Acesso em: 18 mai. 2017.

MONO. *Mono Project*. [S.l.], 2017. Disponível em: <<http://www.mono-project.com/>>. Acesso em: 18 mai. 2017.

MONTGOMERY, J. F.; BEKEY, G. A. Learning helicopter control through "teaching by showing". In: *IEEE. Decision and Control, 1998. Proceedings of the 37th IEEE Conference on*. [S.l.], 1998. p. 3647–3652.

NERIS, L. d. O. *Um piloto automático para as aeronaves do projeto ARARA*. Tese (Doutorado) — Universidade de São Paulo, 2001.

OPEN SOURCE INITIATIVE. *History of the OSI*. [S.l.], 1998. Disponível em: <https://opensource.org/history>. Acesso em: 16 mai. 2017.

QGROUNDCONTROL. *QGroundControl Overview*. [S.l.], 2017. Disponível em: <https://docs.qgroundcontrol.com/en/>. Acesso em: 13 mai. 2017.

RAMOS, H. F. Aeronaves remotamente pilotadas como efeito multiplicador de forças na manutenção da soberania nacional: Popularização da ferramenta enquanto agente transformador do cenário geopolítico. *N/A*, 2014.

SITL. *SITL Simulator*. [S.l.], 2016. Disponível em: <http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html#sitl-simulator-software-in-the-loop>. Acesso em: 16 mai. 2017.

SOUZA, G. d. Análise da viabilidade do uso de vant para mapeamentos topográfico e de cobertura e uso da terra. 2015.

STALLMAN, R. *Why "Open Source" misses the point of Free Software*. [S.l.], 2007. Disponível em: <https://www.gnu.org/philosophy/open-source-misses-the-point.html>.

TOWER. *Repositorio Tower Ground Station Control*. 2016. Disponível em: <https://github.com/DroidPlanner/Tower>. Acesso em: 21 mai. 2017.

VISUAL STUDIO. *Visual Studio Home*. [S.l.], 2017. Disponível em: <https://www.visualstudio.com/>. Acesso em: 16 mai. 2017.

ANEXO A – CÓDIGO-FONTE

KEYBOARDSETUP.CS

Neste anexo encontra-se a classe responsável pela geração da interface que irá guiar o usuário na configuração do sistema de controle do VANT pelo teclado.

```
using System;
using System.Windows.Forms;
using System.Runtime.InteropServices;
using MissionPlanner.Utilities;

namespace MissionPlanner.Keyboard
{
    public partial class KeyboardSetup : Form
    {
        [DllImport("user32.dll")]
        static extern bool HideCaret(IntPtr hWnd);

        globalKeyboardHook gkh =
            globalKeyboardHook.UniqueInstance;

        public KeyboardSetup()
        {
            InitializeComponent();
            MissionPlanner.Utilities.
                Tracking.AddPage(this.GetType().ToString(),
                    this.Text);
        }

        public void Keyboard_Load(object sender, EventArgs e)
        {
            //fixed border
            this.FormBorderStyle = FormBorderStyle.FixedSingle;
            //maximize button removed
            this.MaximizeBox = false;
        }
    }
}
```

```

setModeComboBox.DataSource =
    Common.getModesList(MainV2.comPort.MAV.cs);
setModeComboBox.ValueMember = "Key";
setModeComboBox.DisplayMember = "Value";

if (MainV2.keyboard != null &&
    (MainV2.keyboard.Enabled))
{
    BUT_enable.Text = "Disable";
}

if (MainV2.comPort.BaseStream.IsOpen)
{
    lblVehicleConnected.Text =
        MainV2.comPort.MAV.aptype.ToString();
}
}

private void BUT_enable_Click(object sender, EventArgs
e)
{
    if (MainV2.keyboard != null &&
        (!MainV2.keyboard.Enabled))
    {
        if (MainV2.comPort.BaseStream.IsOpen)
        {
            lblVehicleConnected.Text =
                MainV2.comPort.MAV.aptype.ToString();
            try
            {
                MainV2.keyboard.Accelerate =
                    (Keys) System.Enum.Parse(typeof(Keys),
                        accelerateBox.Text, true);
                MainV2.keyboard.Decelerate =
                    (Keys) System.Enum.Parse(typeof(Keys),
                        decelerateBox.Text, true);
                MainV2.keyboard.RollLeft =
                    (Keys) System.Enum.Parse(typeof(Keys),
                        rollLeftBox.Text, true);
            }

```

```
MainV2.keyboard.RollRight =
    (Keys) System.Enum.Parse(typeof(Keys),
        rollRightBox.Text, true);
MainV2.keyboard.SteerLeft =
    (Keys) System.Enum.Parse(typeof(Keys),
        steerLeftBox.Text, true);
MainV2.keyboard.SteerRight =
    (Keys) System.Enum.Parse(typeof(Keys),
        steerRightBox.Text, true);
MainV2.keyboard.PitchForward =
    (Keys) System.Enum.Parse(typeof(Keys),
        pitchForwardBox.Text, true);
MainV2.keyboard.PitchBackward =
    (Keys) System.Enum.Parse(typeof(Keys),
        pitchBackwardBox.Text, true);
MainV2.keyboard.Arm =
    (Keys) System.Enum.Parse(typeof(Keys),
        armBox.Text, true);
MainV2.keyboard.Desarm =
    (Keys) System.Enum.Parse(typeof(Keys),
        desarmBox.Text, true);
MainV2.keyboard.SetModeKey =
    (Keys) System.Enum.Parse(typeof(Keys),
        setModeBox.Text, true);
MainV2.keyboard.SetMode =
    setModeComboBox.Text;
MainV2.keyboard.PitchValue =
    pitchTrackBar.Value;
MainV2.keyboard.YawValue =
    yawTrackBar.Value;
MainV2.keyboard.RollValue =
    rollTrackBar.Value;
MainV2.keyboard.ThrottleValue =
    throttleTrackBar.Value;
}
catch
{
    MainV2.instance.Invoke((System.Action)
        delegate
```

```
        {
            CustomMessageBox.Show("Please
                                   insert a key in all boxes
                                   before pressing enable", "Empty
                                   Boxes");
        });
        return;
    }
    MainV2.keyboard.Hook();
    MainV2.keyboard.Enabled = true;
    BUT_enable.Text = "Disable";
    timer1.Start();

}
else
    CustomMessageBox.Show("Please connect a
                           UAV first", "Open ComPort");
}
else
{
    MainV2.keyboard.Unhook();
    MainV2.keyboard.Enabled = false;
    timer1.Stop();
    BUT_enable.Text = "Enable";
}
}

private void timer1_Tick(object sender, EventArgs e)
{
    if (MainV2.keyboard != null)
    {
        progressBarRoll.Value =
            MainV2.comPort.MAV.cs.rcoverridech1;
        progressBarPitch.Value =
            MainV2.comPort.MAV.cs.rcoverridech2;
        progressBarThrottle.Value =
            MainV2.comPort.MAV.cs.rcoverridech3;
        progressBarYaw.Value =
            MainV2.comPort.MAV.cs.rcoverridech4;
```

```
try
{
    progressBarRoll.Maximum =
        MainV2.keyboard.checkChannel(1, "max");
    progressBarPitch.Maximum =
        MainV2.keyboard.checkChannel(2, "max");
    progressBarThrottle.Maximum =
        MainV2.keyboard.checkChannel(3, "max");
    progressBarYaw.Maximum =
        MainV2.keyboard.checkChannel(4, "max");

    progressBarRoll.Minimum =
        MainV2.keyboard.checkChannel(1, "min");
    progressBarPitch.Minimum =
        MainV2.keyboard.checkChannel(2, "min");
    progressBarThrottle.Minimum =
        MainV2.keyboard.checkChannel(3, "min");
    progressBarYaw.Minimum =
        MainV2.keyboard.checkChannel(4, "min");
}
catch
{
    //Exception Error in the application.
    -2147024866 (DIERR_INPUTLOST)
}

try
{
    rollTrackBar.Maximum =
        (MainV2.keyboard.checkChannel(1, "max")
        - MainV2.keyboard.checkChannel(1,
        "min")) / 2;
    pitchTrackBar.Maximum =
        (MainV2.keyboard.checkChannel(2, "max")
        - MainV2.keyboard.checkChannel(2,
        "min")) / 2;
```

```
        yawTrackBar.Maximum =
            (MainV2.keyboard.checkChannel(4, "max")
            - MainV2.keyboard.checkChannel(4,
            "min")) / 2;
        throttleTrackBar.Maximum =
            (MainV2.keyboard.checkChannel(3, "max")
            - MainV2.keyboard.checkChannel(3,
            "min")) / 2;
        rollTrackBarMaxValue.Text =
            rollTrackBar.Maximum.ToString();
        pitchTrackBarMaxValue.Text =
            pitchTrackBar.Maximum.ToString();
        yawTrackBarMaxValue.Text =
            yawTrackBar.Maximum.ToString();
        throttleTrackBarMaxValue.Text =
            throttleTrackBar.Maximum.ToString();

        rollTrackBar.Minimum = 100;
        pitchTrackBar.Minimum = 100;
        yawTrackBar.Minimum = 100;
        throttleTrackBar.Minimum = 100;
        rollTrackBarMinValue.Text =
            rollTrackBar.Minimum.ToString();
        pitchTrackBarMinValue.Text =
            pitchTrackBar.Minimum.ToString();
        yawTrackBarMinValue.Text =
            yawTrackBar.Minimum.ToString();
        throttleTrackBarMinValue.Text =
            throttleTrackBar.Minimum.ToString();

    }
    catch
    {
        //Exception Error in the application.
        -2147024866 (DIERR_INPUTLOST)
    }
}
}
```

```
string auxText;

public void allTextBox_Click(object sender, EventArgs
    e)
{
    TextBox tbox = (TextBox)sender;
    string auxText = tbox.Text;
    if (MainV2.keyboard != null)
    {
        tbox.Cursor = Cursors.Default;
        if (tbox.ReadOnly)
        {
            tbox.ReadOnly = false;
            tbox.Text = string.Empty;
            tbox.BackColor = TextBox.DefaultBackColor;
            tbox.ForeColor = TextBox.DefaultForeColor;
        }
    }
    HideCaret(tbox.Handle);
}

private void allTextBox_KeyDown(object sender,
    KeyEventArgs e)
{
    TextBox tbox = (TextBox)sender;
    if (!tbox.ReadOnly)
    {
        switch (e.KeyCode)
        {
            case Keys.ControlKey:
                CustomMessageBox.Show("Control Key not
                    allowed.", "Warning");
                tbox.Text = auxText;
                break;
            case Keys.LControlKey:
                CustomMessageBox.Show("Control Key not
                    allowed.", "Warning");
```

```
        tbox.Text = auxText;
        break;
    case Keys.Escape:
        tbox.Text = auxText;
        break;
    default:
        tbox.AppendText(e.KeyCode.ToString());
        break;
    }
    tbox.BackColor = ThemeManager.ControlBGColor;
    tbox.ForeColor = ThemeManager.TextColor;
    tbox.ReadOnly = true;
}
}

private void allTextBox_Leave(object sender, EventArgs
e)
{
    TextBox tbox = (TextBox)sender;
    if (!tbox.ReadOnly)
    {
        tbox.Text = auxText;
        tbox.BackColor = ThemeManager.ControlBGColor;
        tbox.ForeColor = ThemeManager.TextColor;
        tbox.ReadOnly = true;
    }
}

private void allTextBox_MouseEnter(object sender,
EventArgs e)
{
    TextBox tbox = (TextBox)sender;
    if (MainV2.keyboard != null &&
        MainV2.keyboard.Enabled)
    {
        tbox.Cursor = Cursors.No;
    }
    if (MainV2.keyboard != null &&
```

```
        !MainV2.keyboard.Enabled)
    {
        tbox.Cursor = Cursors.Hand;
    }
}

private void BUT_help_Click(object sender, EventArgs e)
{
    new Keyboard_Help().ShowDialog();
}

}

}
```


ANEXO B – CÓDIGO-FONTE

KEYBOARD.CS

Classe responsável por armazenar os dados do usuário como também por realizar uma resposta ao pressionamento de uma tecla. Para conter os dados, a classe necessita ser construída de modo a possuir todas as variáveis que serão utilizadas pelo usuário.

```
using System;
using System.Windows.Forms;

namespace MissionPlanner.Keyboard
{
    public class Keyboard
    {
        private globalKeyboardHook gkh;
        private string setMode;
        private bool enabled = false;

        private float rollValue;
        private float throttleValue;
        private float pitchValue;
        private float yawValue;

        private Keys accelerate;
        private Keys decelerate;
        private Keys rollLeft;
        private Keys rollRight;
        private Keys steerLeft;
        private Keys steerRight;
        private Keys pitchForward;
        private Keys pitchBackward;
        private Keys arm;
        private Keys desarm;
        private Keys setModeKey;

        public Keyboard()
        {
```

```
gkh = globalKeyboardHook.UniqueInstance;
gkh.HookedKeys.Add(Keys.Control);
}

private void gkh_KeyUp(object sender, KeyEventArgs e)
{

    if (e.KeyCode == rollLeft)
    {
        MainV2.comPort.MAV.cs.rcoverridech1 =
            checkChannel(1, "trim");
    }
    if (e.KeyCode == rollRight)
    {
        MainV2.comPort.MAV.cs.rcoverridech1 =
            checkChannel(1, "trim");
    }
    if (e.KeyCode == pitchForward)
    {
        MainV2.comPort.MAV.cs.rcoverridech2 =
            checkChannel(2, "trim");
    }
    if (e.KeyCode == pitchBackward)
    {
        MainV2.comPort.MAV.cs.rcoverridech2 =
            checkChannel(2, "trim");
    }
    if (e.KeyCode == steerLeft)
    {
        MainV2.comPort.MAV.cs.rcoverridech4 =
            checkChannel(4, "trim");
    }
    if (e.KeyCode == steerRight)
    {
        MainV2.comPort.MAV.cs.rcoverridech4 =
            checkChannel(4, "trim");
    }
    if (MainV2.comPort.MAV.aptype ==
        MAVLink.MAV_TYPE.GROUND_ROVER)
```

```
{
    if (e.KeyCode == accelerate)
    {
        MainV2.comPort.MAV.cs.rcoverridech3 =
            checkChannel(3, "trim");
    }
    if (e.KeyCode == decelerate)
    {
        MainV2.comPort.MAV.cs.rcoverridech3 =
            checkChannel(3, "trim");
    }
}
e.Handled = true;
}

private void gkh_KeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyCode == arm)
    {
        MainV2.instance.BeginInvoke((System.Windows.Forms.MethodInvoker)
            ()
            {
                try
                {
                    MainV2.comPort.doARM(true);
                }
                catch { CustomMessageBox.Show("Failed to
                    Arm"); }
            });
    }
    if (e.KeyCode == desarm)
    {
        MainV2.instance.BeginInvoke((System.Windows.Forms.MethodInvoker)
            ()
            {
                try
                {
                    MainV2.comPort.doARM(false);
                }
            });
    }
}
```

```
        catch { CustomMessageBox.Show("Failed to
            Disarm"); }
    });
}
if (e.KeyCode == setModeKey)
{
    MainV2.instance.BeginInvoke((System.Windows.Forms.MethodInv
        ())
    {
        try
        {
            MainV2.comPort.setMode(setMode);
        }
        catch { CustomMessageBox.Show("Failed to
            change Modes"); }
    });
}
if (e.KeyCode == rollLeft)
{
    if (Control.ModifierKeys == Keys.Control)
        MainV2.comPort.MAV.cs.rcoverridech1 =
            checkChannel(1, "min");
    else
        MainV2.comPort.MAV.cs.rcoverridech1 =
            (ushort)(checkChannel(1, "trim") -
                Convert.ToUInt16(rollValue));
}
if (e.KeyCode == rollRight)
{
    if (Control.ModifierKeys == Keys.Control)
        MainV2.comPort.MAV.cs.rcoverridech1 =
            checkChannel(1, "max");
    else
        MainV2.comPort.MAV.cs.rcoverridech1 =
            (ushort)(checkChannel(1, "trim") +
                Convert.ToUInt16(rollValue));
}
if (e.KeyCode == pitchForward)
{

```

```
    if (Control.ModifierKeys == Keys.Control)
        MainV2.comPort.MAV.cs.rcoverridech2 =
            checkChannel(2, "min");
    else
        MainV2.comPort.MAV.cs.rcoverridech2 =
            (ushort) (checkChannel(2, "trim") -
                Convert.ToUInt16(pitchValue));
}
if (e.KeyCode == pitchBackward)
{
    if (Control.ModifierKeys == Keys.Control)
        MainV2.comPort.MAV.cs.rcoverridech2 =
            checkChannel(2, "max");
    else
        MainV2.comPort.MAV.cs.rcoverridech2 =
            (ushort) (checkChannel(2, "trim") +
                Convert.ToUInt16(pitchValue));
}
if (MainV2.comPort.MAV.aptype ==
    MAVLink.MAV_TYPE.GROUND_ROVER)
{
    if (e.KeyCode == accelerate)
    {
        if (Control.ModifierKeys == Keys.Control)
            MainV2.comPort.MAV.cs.rcoverridech3 =
                checkChannel(3, "min");
        else
            MainV2.comPort.MAV.cs.rcoverridech3 =
                (ushort) (checkChannel(3, "trim") -
                    Convert.ToUInt16(throttleValue));
    }
    if (e.KeyCode == decelerate)
    {
        if (Control.ModifierKeys == Keys.Control)
            MainV2.comPort.MAV.cs.rcoverridech3 =
                checkChannel(3, "max");
        else
            MainV2.comPort.MAV.cs.rcoverridech3 =
                (ushort) (checkChannel(3, "trim") +
```

```
        Convert.ToUInt16(throttleValue));
    }

}

else
{
    if (e.KeyCode == accelerate)
    {
        if (MainV2.comPort.MAV.cs.rcoverridech3 <
            checkChannel(3, "max"))
            MainV2.comPort.MAV.cs.rcoverridech3 +=
                (ushort)(100);
    }
    if (e.KeyCode == decelerate)
    {
        if (MainV2.comPort.MAV.cs.rcoverridech3 >
            checkChannel(3, "min"))
            MainV2.comPort.MAV.cs.rcoverridech3 -=
                (ushort)(100);
    }
}

if (e.KeyCode == steerLeft)
{
    if (Control.ModifierKeys == Keys.Control)
        MainV2.comPort.MAV.cs.rcoverridech4 =
            checkChannel(4, "min");
    else
        MainV2.comPort.MAV.cs.rcoverridech4 =
            (ushort)(checkChannel(4, "trim") -
                Convert.ToUInt16(yawValue));
}

if (e.KeyCode == steerRight)
{
    if (Control.ModifierKeys == Keys.Control)
        MainV2.comPort.MAV.cs.rcoverridech4 =
            checkChannel(4, "max");
    else
        MainV2.comPort.MAV.cs.rcoverridech4 =
```

```

        (ushort) (checkChannel(4, "trim") +
        Convert.ToInt16(yawValue));
    }
    e.Handled = true;
}

private void clearRCOverride()
{
    MAVLink.mavlink_rc_channels_override_t rc = new
        MAVLink.mavlink_rc_channels_override_t();

    rc.target_component = MainV2.comPort.MAV.compid;
    rc.target_system = MainV2.comPort.MAV.sysid;

    rc.chan1_raw = 0;
    rc.chan2_raw = 0;
    rc.chan3_raw = 0;
    rc.chan4_raw = 0;
    rc.chan5_raw = 0;
    rc.chan6_raw = 0;
    rc.chan7_raw = 0;
    rc.chan8_raw = 0;

    try
    {
        MainV2.comPort.sendPacket(rc,
            rc.target_system, rc.target_component);
        MainV2.comPort.sendPacket(rc,
            rc.target_system, rc.target_component);
        MainV2.comPort.sendPacket(rc,
            rc.target_system, rc.target_component);
    }
    catch { }
}

public ushort checkChannel(int chan, string minmaxtrim)
{
    ushort min, max, trim = 0;

```

```
if (MainV2.comPort.MAV.param.Count > 0)
{
    try
    {
        if
        (MainV2.comPort.MAV.param.ContainsKey("RC"
        + chan + "_MIN"))
        {
            min =
                (ushort)(float)(MainV2.comPort.MAV.param["RC"
                + chan + "_MIN"]);
            max =
                (ushort)(float)(MainV2.comPort.MAV.param["RC"
                + chan + "_MAX"]);
            trim =
                (ushort)(float)(MainV2.comPort.MAV.param["RC"
                + chan + "_TRIM"]);
        }
        else
        {
            min = 1000;
            max = 2000;
            trim = 1500;
        }
    }
    catch
    {
        min = 1000;
        max = 2000;
        trim = 1500;
    }
}
else
{
    min = 1000;
    max = 2000;
    trim = 1500;
}
if (minmaxtrim == "min")
```

```
        return min;
    if (minmaxtrim == "max")
        return max;
    if (minmaxtrim == "trim")
        return trim;
    else
        return 0;
}

public bool Enabled
{
    get
    {
        return enabled;
    }
    set
    {
        enabled = value;
        if (enabled)
        {
            MainV2.comPort.MAV.cs.rcoverridech1 =
                checkChannel(1, "trim");
            MainV2.comPort.MAV.cs.rcoverridech2 =
                checkChannel(2, "trim");
            MainV2.comPort.MAV.cs.rcoverridech3 =
                checkChannel(3, "min");
            MainV2.comPort.MAV.cs.rcoverridech4 =
                checkChannel(4, "trim");
        }
        else
        {
            clearRCOverride();
        }
    }
}

public Keys Accelerate {
```

```
        get
        {
            return accelerate;
        }
        set
        {
            gkh.HookedKeys.Add(value);
            accelerate = value;
        }
    }

    public Keys Decelerate {
        get
        {
            return decelerate;
        }
        set
        {
            gkh.HookedKeys.Add(value);
            decelerate = value;
        }
    }

    public Keys RollLeft {
        get
        {
            return rollLeft;
        }
        set
        {
            gkh.HookedKeys.Add(value);
            rollLeft = value;
        }
    }

    public Keys RollRight {
```

```
        get
        {
            return rollRight;
        }
        set
        {
            gkh.HookedKeys.Add(value);
            rollRight = value;
        }
    }

    public Keys SteerLeft {
        get
        {
            return steerLeft;
        }
        set
        {
            gkh.HookedKeys.Add(value);
            steerLeft = value;
        }
    }

    public Keys SteerRight {
        get
        {
            return steerRight;
        }
        set
        {
            gkh.HookedKeys.Add(value);
            steerRight = value;
        }
    }

    public Keys PitchForward
```

```
{
    get
    {
        return pitchForward;
    }
    set
    {
        gkh.HookedKeys.Add(value);
        pitchForward = value;
    }
}

public Keys PitchBackward
{
    get
    {
        return pitchBackward;
    }
    set
    {
        gkh.HookedKeys.Add(value);
        pitchBackward = value;
    }
}

public Keys Arm
{
    get
    {
        return arm;
    }
    set
    {
        gkh.HookedKeys.Add(value);
        arm = value;
    }
}
```

```
}

public Keys Desarm
{
    get
    {
        return desarm;
    }
    set
    {
        gkh.HookedKeys.Add(value);
        desarm = value;
    }
}

public Keys SetModeKey
{
    get
    {
        return setModeKey;
    }
    set
    {
        gkh.HookedKeys.Add(value);
        setModeKey = value;
    }
}

public float RollValue
{
    get
    {
        return rollValue;
    }

    set
    {
```

```
        rollValue = value;
    }
}

public float ThrottleValue
{
    get
    {
        return throttleValue;
    }

    set
    {
        throttleValue = value;
    }
}

public float PitchValue
{
    get
    {
        return pitchValue;
    }

    set
    {
        pitchValue = value;
    }
}

public float YawValue
{
    get
    {
        return yawValue;
    }

    set
    {
```

```
        yawValue = value;
    }
}

public string SetMode
{
    get
    {
        return setMode;
    }

    set
    {
        setMode = value;
    }
}

public void Hook()
{
    gkh.KeyDown += new KeyEventHandler(gkh_KeyDown);
    gkh.KeyUp += new KeyEventHandler(gkh_KeyUp);
    gkh.hook();
}

public void Unhook()
{
    gkh.KeyDown -= new KeyEventHandler(gkh_KeyDown);
    gkh.KeyUp -= new KeyEventHandler(gkh_KeyUp);
    gkh.HookedKeys.Clear();
    gkh.unhook();
}
}
```


ANEXO C – CÓDIGO-FONTE

THREAD DO TECLADO

Thread responsável por verificar se o teclado está habilitado. Caso esteja, o código verifica se há algum veículo conectado a aplicação e envia os dados sobrescritos pelo teclado aos respectivos canais do drone. Este código é inicializado no momento da abertura da aplicação.

```
private void keyboardsend()
{
    float rate = 50; // 1000 / 50 = 20 hz
    int count = 0;

    DateTime lastratechange = DateTime.Now;

    keyboardthreadrun = true;

    while (keyboardthreadrun)
    {
        keyboardsendThreadExited = false;
        try
        {
            if (keyboard)
            {
                MAVLink.mavlink_rc_channels_override_t rc
                    = new
                        MAVLink.mavlink_rc_channels_override_t();

                rc.target_component = comPort.MAV.compid;
                rc.target_system = comPort.MAV.sysid;

                if (MainV2.comPort.MAV.cs.rcoverridech1 !=
                    0)
                {
                    rc.chan1_raw =
                        MainV2.comPort.MAV.cs.rcoverridech1;
                }
                if (MainV2.comPort.MAV.cs.rcoverridech2 !=
                    0)
                {
                    rc.chan2_raw =
```

```
        MainV2.comPort.MAV.cs.rcoverridech2;
    if (MainV2.comPort.MAV.cs.rcoverridech3 !=
        0)
        rc.chan3_raw =
            MainV2.comPort.MAV.cs.rcoverridech3;
    if (MainV2.comPort.MAV.cs.rcoverridech4 !=
        0)
        rc.chan4_raw =
            MainV2.comPort.MAV.cs.rcoverridech4;
    if (MainV2.comPort.MAV.cs.rcoverridech5 !=
        0)
        rc.chan5_raw =
            MainV2.comPort.MAV.cs.rcoverridech5;
    if (MainV2.comPort.MAV.cs.rcoverridech6 !=
        0)
        rc.chan6_raw =
            MainV2.comPort.MAV.cs.rcoverridech6;
    if (MainV2.comPort.MAV.cs.rcoverridech7 !=
        0)
        rc.chan7_raw =
            MainV2.comPort.MAV.cs.rcoverridech7;
    if (MainV2.comPort.MAV.cs.rcoverridech8 !=
        0)
        rc.chan8_raw =
            MainV2.comPort.MAV.cs.rcoverridech8;

    if (lastkeyboard.AddMilliseconds(rate) <
        DateTime.Now)
    {
        if (!comPort.BaseStream.IsOpen)
            continue;

        if (comPort.BaseStream.BytesToWrite <
            50)
        {
            if (sit1)
            {
                MissionPlanner.Controls.SITL.rcinput();
            }
        }
    }
}
```

```
        else
        {
            comPort.sendPacket(rc);
        }
        count++;
        lastkeyboard = DateTime.Now;
    }

    }

    Thread.Sleep(20);
}
catch
{
}
}
```