UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

Eugênio Piveta Pozzobon

DESENVOLVIMENTO DE UMA PLATAFORMA EQUATORIAL DE BAIXO CUSTO PARA ASTROFOTOGRAFIA

Santa Maria, RS
2022

Eugênio Piveta Pozzobon

DESENVOLVIMENTO DE UMA PLATAFORMA EQUATORIAL DE BAIXO CUSTO PARA ASTROFOTOGRAFIA

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Engenharia de Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Engenharia de Computação**.

ORIENTADOR: Prof. Rafael Concatto Beltrame

Santa Maria, RS
2022

Eugênio Piveta Pozzobon

DESENVOLVIMENTO DE UMA PLATAFORMA EQUATORIAL DE BAIXO CUSTO PARA ASTROFOTOGRAFIA

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Engenharia de Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Engenharia de Computação**.

Aprovado em 00 de de 2022:

Rafael Concatto Beltrame, Dr. (UFSM)
(Presidente/Orientador)

Santa Maria, RS
2022

RESUMO

DESENVOLVIMENTO DE UMA PLATAFORMA EQUATORIAL DE BAIXO CUSTO PARA ASTROFOTOGRAFIA

AUTOR: Eugênio Piveta Pozzobon
ORIENTADOR: Rafael Concatto Beltrame

Atualmente, a astrofotografia é realizada por telescópios monumentais e também por um grande número de astrofotógrafos amadores, e estes contam com inúmeros desafios. O principal deles reside no fato que corpos celestes, de forma geral, demandam elevados tempos de exposição. Infelizmente, caso a câmera esteja imóvel sobre um tripé, o movimento de rotação da Terra não permite que os astros sejam expostos ao sensor por muito tempo, pois as imagens acabam borradas. Por esse motivo, é necessário o uso de uma ferramenta que movimente a câmera no sentido de rotação aparente do céu, compensando esse movimento e obtendo-se um registro fotográfico de alta qualidade. Para isso, existem inúmeras ferramentas comerciais para o rastreamento do céu, porém, todas elas são comercializadas no hemisfério Norte e com um custo excessivo para o brasileiro médio. Então, a fim de simplificar e reduzir o custo associado à astrofotografia, tornando-a acessível, objetivou-se com este trabalho desenvolver uma plataforma equatorial para astrofotografia que seja portátil; robusta; precisa; de fácil configuração e utilização; de peso e volume compatíveis com tripés fotográficos; e com custo inferior à soluções comerciais. A plataforma tem como diferencial um aplicativo mobile que descomplica o uso dessas ferramentas para configuração e setup da plataforma para a obtenção de registros fotográficos. O sistema final passou por testes de vibração para garantir que a montagem estaria adequada para o uso, e também com testes em campo, onde foram tiradas fotografias da via-láctea, comparando-se resultados fotográficos com e sem o uso da plataforma desenvolvida.

Palavras-chave: Astrofotografia. Plataforma Equatorial. Sistema Eletrônico. Aplicativo Android. Bluetooth.

ABSTRACT

DESENVOLVIMENTO DE UMA PLATAFORMA EQUATORIAL DE BAIXO CUSTO PARA ASTROFOTOGRAFIA

AUTHOR: Eugênio Piveta Pozzobon
ADVISOR: Rafael Concatto Beltrame

Currently, astrophotography remains practiced by incredible telescopes and also by astrophotographers that have countless challenges. The main one relies on the fact that celestial bodies, in general, demand long exposure times. Unfortunately, despite the camera being on a tripod, the rotational movement of the Earth doesn't allow the stars to be exposed to the sensor for a long time, resulting in blurry images. For this reason, it is necessary to use a tool that moves the camera in the apparent rotation direction of the sky, compensating for this movement and obtaining a high-quality photographic record. For this, there are numerous commercial tools for tracking the sky. However, all of them are commercialized in the Northern hemisphere and with an exorbitant cost for the average Brazilian. So, in order to simplify and reduce the cost associated with astrophotography, making it accessible, the objective of this work was to develop an equatorial platform for astrophotography that is portable; robust; precise; easy to set up and use; of weight and volume compatible with photographic tripods; and at a lower cost than commercial solutions. The platform's differential is a mobile application that makes it easy to use these tools for the configuration and setup of the platform to obtain photographic records. The final system has been passed into vibration tests to ensure that the assembly would be suitable for use. Also, the system got tested with long exposure photography, comparing photographic results with and without using the developed platform.

Keywords: Astrophotography. Equatorial Mount. Electronic System. Android Application. Bluetooth.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇAO

Desde os primeiros registros datados, civilizações já observavam o céu e, desse modo, podiam contabilizar a passagem do tempo, identificar as estações do ano, os ciclos sazonais de chuvas e/ou secas, etc. Assim, por exemplo, era possível realizar um planejamento mais assertivo acerca da melhor época de plantio e colheita de diferentes culturas. Cada civilização tinha seu meio e técnica de observação. Por exemplo, em 4000 a.C., os povos da Mesopotâmia utilizavam zigurates para observar o céu noturno ; já em 2500 a.C., foi construída a estrutura de pedras Stonehenge, na Inglaterra, para registrar os solstícios. Foi somente em 1609 d.C. que Galileu conseguiu aperfeiçoar e utilizar um telescópio refrator para observar os planetas e as estrelas pela primeira vez. (HELERBROCK, Rafael, c2021)

A astrofotografia consiste no emprego de técnicas fotográficas para registrar objetos astronômicos, como planetas, estrelas, galáxias, nebulosas, etc. A primeira é datada de 1840 e é um registro da Lua. (CABAU JÚNIOR, S. D., 2016) Desse período em diante, a fotografia teve um papel muito importante na observação celeste e no registro do céu para análise científica. Essa técnica foi evoluindo gradualmente de forma que, por volta de 1960, já havia equipamentos que permitiam realizar registros mais concretos e eficientes, possibilitando fotos com mais definição e nitidez. (SANTOS, N. S., 2010)

No fim do século XX, telescópios maiores e mais complexos, instalados na Terra ou a orbitando no espaço (como o telescópio espacial Hubble), ampliaram a capacidade da ciência em estudar fenômenos astronômicos ou mesmo a origem do próprio universo. (SANTOS, N. S., 2010) Paralelamente, as ferramentas amadoras para astronomia (como telescópios de baixo custo), ou mesmo para astrofotografia (câmeras de custo mais acessível e equipamentos para rastreamento do céu) continuaram a ser desenvolvidas, de forma que um grande número de astrônomos e, especificamente, astrofotógrafos amadores continuassem exercendo seu hobby ou mesmo contribuindo à ciência.

Nesse sentido, o desenvolvimento de equipamentos acessíveis voltados ao público amador tem um papel fundamental para despertar o interesse pela ciência em cada vez mais pessoas. Atualmente, o custo de muitas ferramentas classificadas como “amadoras” ainda é elevado, principalmente para a realidade brasileira. Como exemplo, hoje podem ser empregados pequenos telescópios comerciais ou artesanais acoplados a câmeras digitais (de celular ou DSLRs (Digital Single Lens Reflex). No entanto, os principais desafios de se fotografar galáxias, nebulosas, etc., é que esses corpos, de forma geral, demandam elevados tempos de exposição.(CABAU JÚNIOR, S. D., 2016) Infelizmente, o movimento de rotação da Terra não permite que os astros sejam expostos ao sensor por muito tempo, pois as imagens ficariam borradas. Por esse motivo, é necessário o uso de uma ferramenta que movimente a câmera (acoplada ou não a um telescópio) no sentido de rotação

aparente do céu, compensando o movimento e permitindo que o sensor da câmera possa receber luz por longos períodos de tempo (de minutos a horas).(CABAU JÚNIOR, S. D., 2016) Assim, consegue-se obter um registro fotográfico de alta qualidade e com um baixo custo computacional de pós processamento.

Nesse sentido, existem inúmeras ferramentas comerciais para o rastramento rastreamento do céu voltadas ao público amadador, como Nyx Track, SkyGuiderTM Pro, PolarieTM, entre outras (conforme Figura 1.1). Porém, todas elas são comercializadas no hemisfério norte e com um custo excessivo para brasileiro médio (considerando taxa de câmbio, taxas de importação e frete). Assim, de modo a contribuir à popularização da astrofotografia no Brasil, incentivando cada vez mais jovens a seguirem na carreira científica, propõe-se o desenvolvimento de uma plataforma equatorial de baixo custo para astrofotografia.

Figura 1.1 – Exemplo de solução comercial empregada em astrofotografias.

Fonte: (iOptron Corporation, c2021)

2 DESENVOLVIMENTO TEÓRICO

2.1 ASTROFOGRAFIA

2.1.1 Comprimento Focal

2.1.2 Exposição

2.1.2.1 Velocidade

2.1.2.2 Abertura

2.1.2.3 ISO

2.1.3 Formatos de Arquivos

2.1.4 Empilhamento de Fotos

Softwares de empilhamento e soluções para pós processamento

2.2 PLATAFORMAS EQUATORIAIS

2.2.1 Métodos de Alinhamento Polar

2.2.1.1 Ajuste de Azimute

2.2.1.1.1 Localização da Estrela Polar

Uso de lunetas para alinhamento

2.2.1.1.2 Alinhamento com o Polo Norte

Declinação Magnética

2.2.1.2 Ajuste de Elevação

2.2.2 Soluções Comerciais Existentes

Existem inúmeras soluções comerciais para o problema proposto, porém todos eles usam a luneta como método de alinhamento polar e isso só é praticável no hemisfério norte devido ao forte brilho da estrela polar diferentemente do situação no hemisfério sul. Existem produtos para todo o tamanho de orçamento. A tabela 2.1 ilustra a concorrência dos principais equipamentos, comparando as principais funcionalidades.

Tabela 2.1 – Comparativo das Soluções de Mercado

	Nyx Tracker	iOptron	Vixen Optics	SkyWatcher
Preço (US$)	115	299	399	299
Carga Máxima (kg)	2.25	3	2	3
Erro periódico (arcsec)	115	100	50	50
Volume (cm^2)	155	490	323	220
Peso (kg)	0,4	1,15	0,79	0,72
Alinhamento	*Laser*	*Polar Scope*	*Polar Scope*	*Polar Scope*

Fonte: (Egan, Mark, c2021)

Contudo, na realidade brasileira, o preço mostrado passaria ainda por impostos, tornando a compra mais inviável. O Nyx Tracker (Figura 2.1) é o sistema mais barato, com alinhamento impraticável no hemisfério sul, e também o mais simples em materiais.

Figura 2.1 – Nyx Tracker

Fonte: (Egan, Mark, c2021)

2.3 OBJETIVOS

Pelo *benchmark* exposto, fixou-se como objetivos do sistema um produto robusto, visualmente elegante, e que consiga se aproximar das propriedades do modelo comercial mais barato, com erro periódico igual ou inferior a 115 arcsec, peso e volume adequados para o tripé usado, mantendo-se estável, sendo o menor possível e com o preço inferior a 115 dólares. Com o diferencial de um aplicativo que permita uma fácil interação do usuário com o sistema, descomplicando o processo.

2.4 PROTOCOLOS DE COMUNICAÇÃO

2.4.1 Serial

2.4.1.1 UART

Velocidade, Falhas de comunicação, Guidelines de Design de PCB

2.4.1.2 I2C

Endereçamento, Velocidade, Guidelines de Design de PCB

2.4.2 Bluetooth

2.5 SENSORES E ATUADORES

2.5.1 Acelerômetro

2.5.2 Giroscópio

2.5.3 Magnetômetro

2.5.4 GPS

2.5.5 Motor de Passo

Driver, formas de Acionamento...

2.6 MICROCONTROLADORES

2.6.1 Arduino Nano

Justificativa, diagrama do Arduíno

2.7 INTERFACE GRÁFICA

2.7.1 Princípios e Diretrizes

Os princípios e as diretrizes comumente utilizados em IHC giram em torno dos seguintes tópicos: correspondência com as expectativas dos usuários; simplicidade nas estruturas das tarefas; equilíbrio entre controle e liberdade do usuário; consistência e padronização; promoção da eficiência do usuário; antecipação das necessidades do usuário; visibilidade e reconhecimento; conteúdo relevante e expressão adequada; e projeto para erros. (BARBOSA et al., 2021) Esse conjunto de princípios são conhecidos como heurística de Nielsen, pois são aplicáveis em qualquer sistema, independente de casos específicos.

2.7.1.1 Visibilidade dos status do sistema

O sistema deve sempre manter o usuário atualizado sobre as condições de operação com uma taxa de atualização condizente para a informação. Ao informar o status da bateria, por exemplo, o usuário do smartphone consegue predizer quanto tempo de uso ainda terá e irá conseguir manejar sua interação com base nessa previsibilidade. (Nielsen, Jakob, 2020)

2.7.1.2 Comunicar-se com o mundo real

O Design tem que se comunicar com o usuário na língua do usuário. Se um brasileiro não sabe inglês, ele ficará perdido nos Estados Unidos, pela mesma lógica, se a máquina não consegue se comunicar com o usuário, então ele ficará perdido.

Da mesma forma, o desenvolvedor não pode assumir que o usuário entenderá o aplicativo somente pelo fato do desenvolvedor ter feito algo que ele próprio entenda; é sempre preciso conferir a linguagem do sistema com um conjunto grande de pessoas para evitar mal entendidos, que se já ocorrem em língua nativa, na língua do sistema só tende a piorar.

Quando o usuário não entende a língua do sistema, ele se sente afastado e irá deixar de usar a plataforma. É interessante que a plataforma tenha designs semelhantes com objetos do mundo real, dessa forma, o usuário se sente "contemplado"e consegue facilmente fazer a conexão entre o mundo real e a plataforma. (Kaley, Anna, 2018)

2.7.1.3 Liberdade de Controle do Usuário

Por vezes, a pessoa que está realizando um processo em um sistema pode cometer um engano. Esse evento pode levar a situações de erro que não devem comprometer a experiência. Por isso, os usuários precisam de uma “saída de emergência” claramente marcada para sair do estado indesejado. Isso reduz a sua ansiedade e o medo de errar, pois ele sabe que os erros podem ser desfeitos. (BARBOSA et al., 2021)

2.7.1.4 Consistências e Padrões

É importante que o sistema mantenha uma consistência entre suas telas, ou mesmo em grandes plataformas, que os múltiplos programas tenham a mesma ’cara’ com funções localizadas no mesmo lugar, com nomes similares e com um disign similar. Exeplo disso é as telas dos aplicativos do google docs; todos possuem o mesmo extilo de menu. MS Office também oferece isso.

A consistência também se extende aos ícones. O vetor que representa um botão, por exemplo, é importante que ele seja consistente em extilo com os demais. Eles podem ser mais preenchidos, mais *clean*, mais neturos, mais suaves. O que importa mais nesse caso é que sejam todos padronizados. (Harley, Aurora, 2014)

2.7.1.5 Prevenção de erros

Uma forma de prevenção é oferecer sugestões numa caixa de pesquisa por exemplo. Em situações de rotina, como disparar um lembrete, a tela de criação pode oferecer uma sugestão defaut de template que faça sentido de fato para o usuário. Para evitar corrupção de dados pelo usuário na hora de um cadastro, é possível ofercer ao usuário que ele preencha números de uma forma truncada, e fazendo um pós processamento para ler o número corretamente.

2.7.1.6 Relembrar o usuário é mais fácil do que o usuário relembrar

Quando o usuário precisa repensar sobre algo incomoum na memória, ele precisa de muito tempo para pensar sobre esse algo. Então, quando a plataforma exige uma lembrança do usuário para entender algo, acaba que isso limita a experiência e incorre em perda de tempo, ou confusão.

Por isso, é mais interessante realizar a exigência com uma possível sugestão de resposta correta. A recognição de algo é muito mais prática para a mente humana pois

ao mostrar para o cérebro algo relacionado com o que precisa lembrar-se, isso dispara a memória de forma mais forte. Dar uma pista para o cérebro é mais eficiente do que simplesmente perguntar sem oferecer nada para a memória. (Budiu, Raluca, 2020)

2.7.1.7 Torne o sistema flexível e eficiente

Atalhos, personalização e customização. Com esses 3 fatores é possível melhorar a usabilidade para aqueles que não são mais novatos no software e isso ajuda a manter esses usuários ativos. Um fotógrafo experiente, que está acostumado com os atalhos de teclado nos aplicativos da Adobe, teria muita "dor de cabeça"se o teclado viesse a falhar, pois a mente já assimilou os atalhos mais usados e eles fazem diferença na velocidade com que o profissional atua com o software. (Laubheimer, Page, 2020)

2.7.1.8 Tenha um Design minimalista

Um design minimalista ajuda a ter somente o que é necessário focar na tela, isso ajuda o usuário a não se sentir perdido. Isso significa usar elementos simples num arranjo onde desenho e a interface combine de forma agradável sem chamar a atenção de forma desnecessária. (Nielsen, Jakob, 2020)

2.7.1.9 Ajude o usuário a entender e se recuperar de erros

O usuário precisa entender quando o sistema não está funcionando bem e como fazê-lo voltar ao normal. As mensagens de erro devem ser expressas em de uma forma simples, indicando o possível problema e a solução. Cores vermelhas e pretas ajudam a demonstrar o sinal de erro para o usuário. (Laubheimer, Page, 2015)

2.7.1.10 Tire dúvidas e documente o sistema

Existem duas formas de ajudar o usuário e tirar suas dúvidas. A primeira é de forma proativa, onde a aplicação guia o usuário para se familiarizar com a interface. Outra forma é por meio de uma seção com perguntas e respotas, essa seção ajuda os usuários a se tornarem mais independetes com a aplicação, resolvendo seus próprios problemas e filtrando os casos que precisam de suporte para a equipe técnica da plataforma. (Joyce, Alita., 2020)

2.7.2 Android

2.7.2.1 Ambiente de Desenvolvimento

O *Android Studio* é o ambiente de desenvolvimento integrado oficial para a criação de apps *Android* e é baseado no *IntelliJ IDEA*. Ele oferece uma série de Recursos que possibilitam a confecção de um aplicativo: Sistema de compilação flexível baseado em *Gradle*; Um emulador rápido com suporte a vários recursos; um ambiente unificado que possibilita o desenvolvimento para qualquer dispositivos Android, incluindo relógios e televisões; Integração com GitHub para backup e documentação do código; entre outras funções que possibilitam análizar o desempenho de um aplicativo em tempo real, bem como fazer updates. (ANDROID. . . , 2021)

2.7.2.2 Linguagens de Programação

Existem soluções de desenvolvimento Android mais *user-friendly* como *APP Inventor* ou *Kodular*, porém, essas interfaces não garantem ao desenvolvedor um pleno controle do aplicativo, e muitas vezes acaba limitando a interface. Por isso, usar linguagens de programação nativas é uma abordagem mais interessante para aplicativos mais completos. É possível criar aplicativos com diversas linguagens, mas somente duas são nativas e permitem realizar aplicações que podem abusar de todo o poder de processamento de um *smartphone*: Java e Kotlin.

Em 2017, Kotlin foi definido pela Google como sendo a principal linguagem de desenvolvimento Android. A linguagem ainda é muito mais nova que Java, sendo desenvolvida em pela JetBrains. As grandes motivação de se usar Kotlin para o desenvolvimento reside nos fatos de ser uma linguagem segura para prevenção de objetos nulos, operando em paralelo com qualquer código em Java e dando opções de co-rotinas. Além disso, ao comparar dois códigos com a mesma função, um escrito em Java, outro em Kotlin; o segundo pode ser até 40% mais compacto, o que implica em uma linguagem mais concisa e compreensível entre desenvolvedores. As desvantagens de se usar Kotlin, para este trabalho, é somente a falta de uma comunidade grande, comparando com Java; o que limita o suporte para eventuais problemáticas de desenvolvimento. (Redka, Maria, 2021)

Dentro do ambiente de Desenvolvimento usa-se também linguagem de arquivos XML para a criação de interfaces gráficas (*layouts*) bem como a escrita dos vetores, animações, e arquivos de configuração do aplicativo e temas de *layout*. Para armazenamento de dados dentro do aplicativo normalmente usa-se uma *database* que é operada com códigos de consulta *SQL*.

2.7.2.3 Material Design

Existem uma série de diretrizes de Design fornecidas pela Google para guiar o desenvolvimento de aplicativos android. Essas informações são fornecidas principalmente pela biblioteca Material Design, que fornece pacotes facilmente implementáveis de layouts para aplicações responsivas e padronizadas.

A Biblioteca colabora com o desenvolvedor fornecendo icones, tipografia, cores e componentes gráficos que trazem uma imersão para o usuário de forma simples e minimalista. Os design se inspiram no mundo real, facilitando a comunicação com o usuário. (MATERIAL. . . , 2015)

3 DESENVOLVIMENTO DA PLATAFORMA

3.1 HARDWARE

3.1.1 Módulos Comerciais

3.1.1.1 Módulo Bluetooth HC05

Justificativa, limitações, pinagem

3.1.1.2 Módulo de sensores GY87

Pinagem

Sensores MPU6050 e HMC

Enderaçamento na rede, velocidades suportadas, filtragem de dados, cálculo dos ângulos.

3.1.2 Motor 28BYJ-48

Justificativa: preço, uso em soluções comerciais.

3.1.2.1 Driver ULN2003

Acionamento escolhido, velocidade e torque

3.1.3 Placa de Circuito Impresso

3.1.3.1 Diagrama Elétrico

3.1.3.2 Layout

3.1.3.3 Manufatura

Foto da placa

3.2 PROJETO ESTRUTURAL

3.2.1 Requisitos de Projeto

Vibração, constância na velocidade, resistência para proteção dos equipamentos.

3.2.2 Peças

3.2.2.1 Frame Principal

3.2.2.2 Engrenagens

Cálculo das dimensões das engrenagens, passo, etc.

3.2.2.3 Barra de Elevação

Dimensionamento

3.2.3 Manufatura

CNC Foto das peças.

3.2.4 Montagem

Observações da montagem, elásticos, etc.

4 DESENVOLVIMENTO DO APLICATIVO

4.1 NECESSIDADES DOS USUÁRIOS

Descrever as necessidades e requisitos mínimos da aplicação. Casos de uso.

4.2 PROTOTIPAÇÃO DAS TELAS

Escrever sobre a prototipação no Adobe XD

4.3 IMPLEMENTAÇÃO DO APLICATIVO ANDROID

4.3.1 Requisitos de Sistema

Requisitos de Sistema, versão mínima do Android, permissões exigidas pelo usuário, etc.

4.3.2 Arquitetura do Funcionamento

4.3.2.1 Activity

4.3.2.2 Room Database

Base de dados criadas e valores armazenados

4.3.2.3 Fragments

4.3.2.3.1 *View Model*

Databinding

4.3.2.4 Protocolo de Comunicação Bluetooth

Diagrama de comunicação, comandos, timeout, checagem de dados, buffer e delay, taxa de atualização e limitações.

5 RESULTADOS OBTIDOS

5.1 ANÁLISE DE VIBRAÇÃO

5.1.1 Método

5.1.2 Discussão

5.2 CUSTO TOTAL DO SISTEMA

5.3 ANÁLISE DE DESEMPENHO DO APLICATIVO

5.4 FOTOGRAFIAS

6 CONSIDERAÇÕES FINAIS

6.1 PROJETO *OPEN-SOURCE*

REFERÊNCIAS BIBLIOGRÁFICAS

ANDROID Studio: Guia do usuário. Google, 2021. Acesso em 16 mar.2021. Disponível em: <https://developer.android.com/studio/intro>.

BARBOSA, S. D. J. et al. **Interação Humano-Computador e Experiência do Usuário**. [S.l.]: Autopublicação, 2021.

Budiu, Raluca. **Memory Recognition and Recall in User Interfaces.** Nielsen Norman Group, 2020. Acessado em 25 de junho de 2021. Disponível em: <https://www.nngroup.com/articles/recognition-and-recall/>.

CABAU JÚNIOR, S. D. **Introdução à astrofotografia**. Network for Astronomy School Education, 2016. Acesso em 20 fev. 2021. Disponível em: <http://sac.csic.es/astrosecundaria/complementario/pt/actividades/instrumentos/astrofotografia.pdf>.

Egan, Mark. **Nyx Tech**. Nyx Tech, c2021. Acesso em 29 mar. 2021. Disponível em: <https://nyxtech.us/>.

Harley, Aurora. **Icon Usability.** Nielsen Norman Group, 2014. Acessado em 25 de junho de 2021. Disponível em: <https://www.nngroup.com/articles/icon-usability/>.

HELERBROCK, Rafael. **História da Astronomia**. Brasil Escola, c2021. Acesso em 20 fev. 2021. Disponível em: <https://brasilescola.uol.com.br/fisica/historia-astronomia.htm.>

iOptron Corporation. **SkyGuiderTM Pro camera mount full package**. IOPTRON, c2021. Acesso em 20 fev. 2021. Disponível em: <https://www.ioptron.com/product-p/3550.htm>.

Joyce, Alita. **Help and Documentation**. Nielsen Norman Group, 2020. Acessado em 25 de junho de 2021. Disponível em: <https://www.nngroup.com/articles/help-and-documentation/>.

Kaley, Anna. **Match Between the System and the Real Worl**. Nielsen Norman Group, 2018. Acessado em 25 de junho de 2021. Disponível em: <https://www.nngroup.com/articles/match-system-real-world/>.

Laubheimer, Page. **Preventing User Errors**. Nielsen Norman Group, 2015. Acessado em 25 de junho de 2021. Disponível em: <https://www.nngroup.com/articles/slips/>.

____. **Flexibility and Efficiency of Use**. Nielsen Norman Group, 2020. Acessado em 25 de junho de 2021. Disponível em: <https://www.nngroup.com/articles/flexibility-efficiency-heuristic/>.

MATERIAL Design: Introduction. Google, 2015. Acesso em 16 mar. 2021. Disponível em: <https://material.io/design/introduction>.

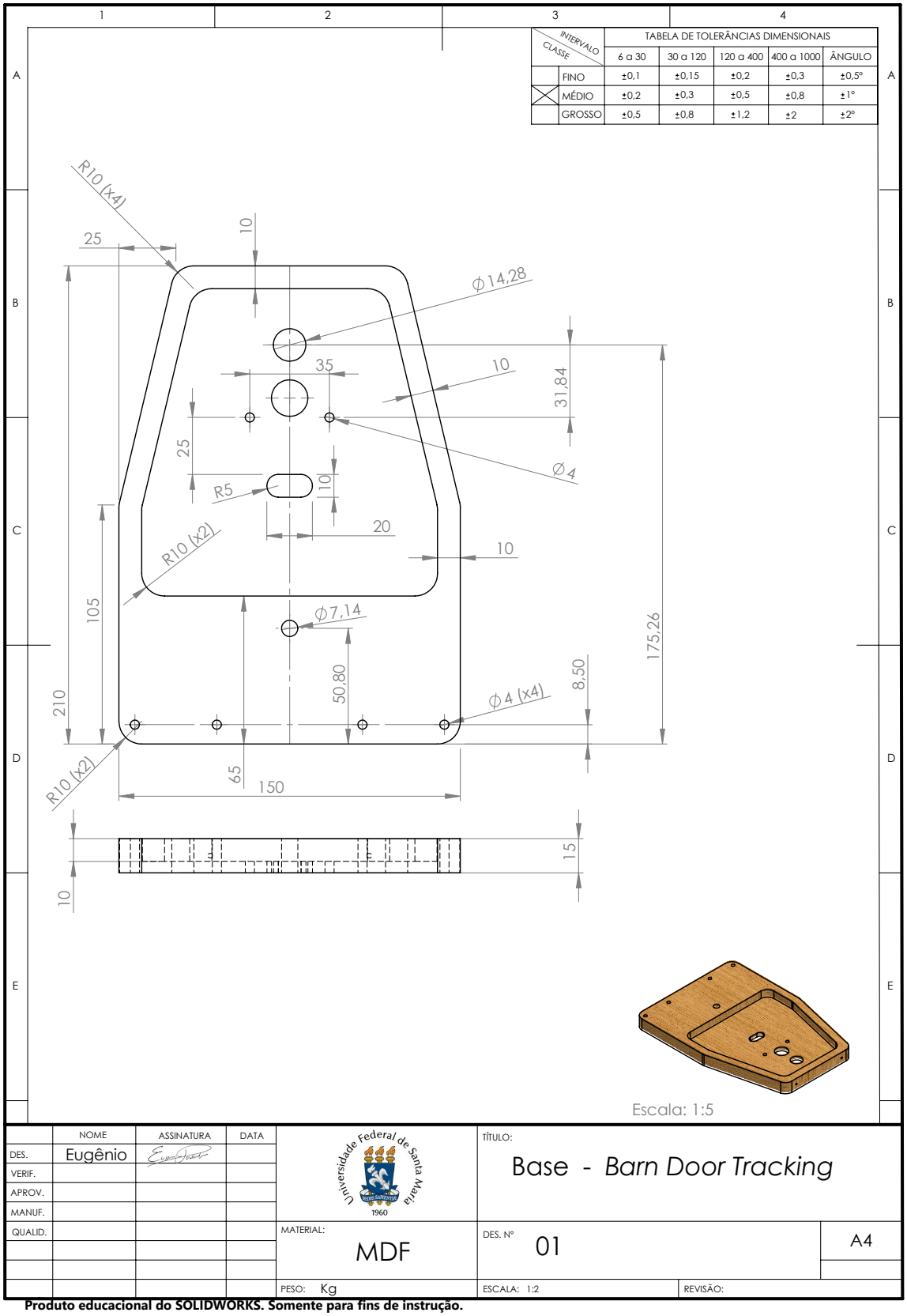
Nielsen, Jakob. **Usability Heuristics for User Interface Design.** Nielsen Norman Group, 2020. Acessado em 25 de junho de 2021. Disponível em: <https://www.nngroup.com/articles/ten-usability-heuristics/>.

Redka, Maria. **Kotlin vs. Java**: Which programming language to choose for your android app. MLSDev, 2021. Acesso em 16 mar. 2021. Disponível em: <https://mlsdev.com/blog/kotlin-vs-java>.

SANTOS, N. S. **A astrofotografia e sua importância para a astronomia**. Educação Espacial, 2010. Acesso em 19 fev. 2021. Disponível em: <https://educacaoespacial.files.wordpress.com/2010/10/a-astrofotografia-e-sua-importncia-para-a-astronomia.pdf>.

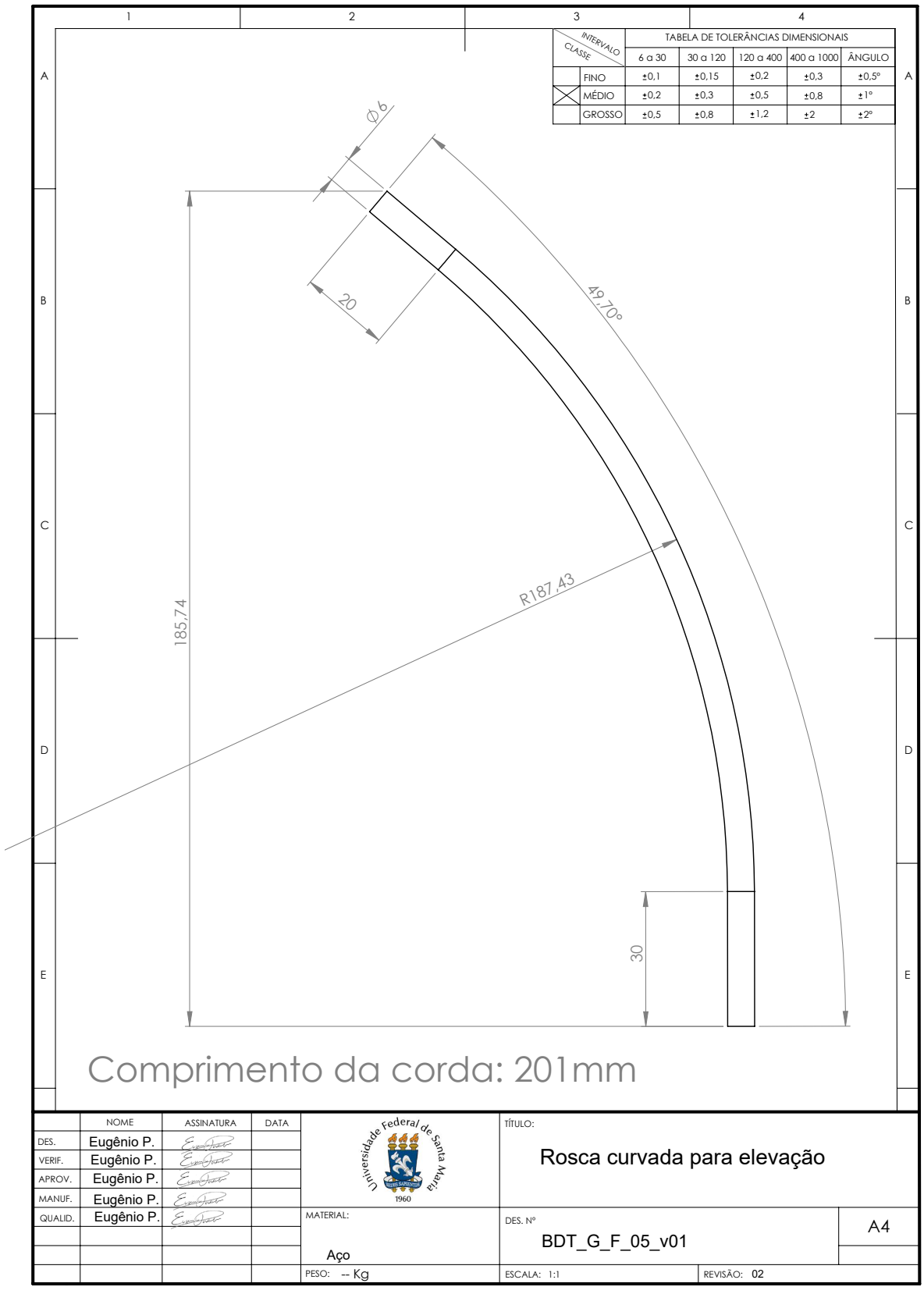
APÊNDICE A – DESENHO TÉCNICO DAS PEÇAS

A.1 – BASE



A.2 – PARTE MÓVEL

A.3 – BARRA CURVADA



APÊNDICE B – DIAGRAMA DE MONTAGEM

APÊNDICE C – CÓDIGO EXCUTADO PELO ARDUINO NANO

```
//Define Variables
double STP_SPEED = 1.89363;//10;//
byte runningSteps = 20;

#define CALIBRATING_TIME 30 //tempo de calibração do sensor magnetômetro.
#define DEBUG

/*
 * Initializate HC05 Serial Comunication
 */
void bluetoothSart() {
  bluetoothSerial.begin(9600);   //Serial Bluetooth
  pinMode(bluetoothStatePin, INPUT); //Enable pin HC05, indicate if it is
   ↪  connected with smartphone or not
}

/*
 * Manages Bluetooth Comunication
 */
void bluetoothCommunication() {

  bool bluetoothState = checkBluetoothCondition();

  if (bluetoothState = true) {

    if ( (bluetoothSerial.available() > 0) || ((millis() -
     ↪  timerBluetoothSendMessages) > 25) ) {//50Hz

      while (bluetoothSerial.available() > 0) {
        timerBluetoothRecieveMessages = millis(); // resset timer to
         ↪  countdown for get ACK response.
        timerBluetoothRestartMessages = millis();
        availableUserOption = bluetoothSerial.read(); //2ms for recieve a
         ↪  single char
        Serial.println(availableUserOption);
      }
```

```
      if ((millis() - timerBluetoothRecieveMessages) < 1000) { //it takes
       ↪  16ms for send all angle buffer
        timerBluetoothSendMessages = millis(); // resset timer to
         ↪  countdown for send message.
        printDataBluetoothSerial(); //send messages to HC05 module
      }

    }

    if ((millis() - timerBluetoothRestartMessages) > 2000) { //if there
     ↪  is no communication with app (ACK) Try restart communication
      timerBluetoothRestartMessages = millis();
      printDataBluetoothSerial(); //send messages to HC05 module
    }

  }

}

void printDataBluetoothSerial() { //it takes 16ms for send this messages
 ↪  buffer
  bluetoothSerial.print(int(angleY * 10)); bluetoothSerial.print(",");
  bluetoothSerial.print(int(angleX * 10)); bluetoothSerial.print(",");
  bluetoothSerial.print(int(angleZ * 10)); bluetoothSerial.print(",");
  bluetoothSerial.print(int(angleX * 10) + int(angleY * 10) + int(angleZ
   ↪  * 10));
  bluetoothSerial.print("\n");
}

void debugSensor() {
#ifdef DEBUG
  Serial.print("\tRoll: ");
  Serial.print(angleY);
  Serial.print("\tPitch: ");
  Serial.print(angleX);
  Serial.print("\tYaw: ");
  Serial.println(angleZ);
#endif
}
void printCalibrationInfo() {
```

```
  #ifdef DEBUG
  Serial.println("Calibration Infos:");
  Serial.print("mxMax: "); Serial.println(mxMax);
  Serial.print("mxMin: "); Serial.println(mxMin);
  Serial.print("myMax: "); Serial.println(myMax);
  Serial.print("myMin: "); Serial.println(myMin);
  Serial.print("mzMax: "); Serial.println(mzMax);
  Serial.print("mzMin: "); Serial.println(mzMin);
#endif
}

void getData() {
  if (((millis() - timerMpuData) >= 0) && !stepperState) {

    timerMpuData = millis();
    //1ms to get all data with 400kHz and 5ms with 100kHz
    accelgyro.readNormalizeAccel();
    accelgyro.readNormalizeGyro();

    mag.getHeading(&mx, &my, &mz);


    angleCalculation();
    //compassCalculation();

    debugSensor();
  }
}

void angleCalculation() { //900us usando int_16t e 1300us usando float
  unsigned long cT = micros(); // contar tempo de loop
  unsigned long dT = cT - gyroIntegrateTimer; //Variable for measure
   ↪  angle with gps and make integration of values
  gyroIntegrateTimer = cT;

  // Gyro Calculations convert gyro raw to degrees per second
  //  rate_gyr_x = accelgyro.ngx;
  //  rate_gyr_y = accelgyro.ngy;
  //  rate_gyr_z = accelgyro.ngz;
  //  gyroXangle += rate_gyr_x * dT;
```

```
  //  gyroYangle += rate_gyr_y * dT;
  //  gyroZangle += rate_gyr_z * dT;

  // Roll angle
  angleX = accelGyroRelation * (-angleX + accelgyro.ngx * (float(dT) /
   ↪  1000000)) + (1 - accelGyroRelation) *
         (atan2(accelgyro.nax, sqrt(pow(accelgyro.nay, 2) +
          ↪  pow(accelgyro.naz, 2))) * 180) / 3.14;
  angleX = - angleX;

  // Pitch angle
  angleY = accelGyroRelation * (-angleY + accelgyro.ngy * (float(dT) /
   ↪  1000000)) + (1 - accelGyroRelation) *
         (atan2(accelgyro.nay, sqrt(pow(accelgyro.nax, 2) +
          ↪  pow(accelgyro.naz, 2))) * 180) / 3.14;
  angleY = - angleY;
  //(atan2(accelgyro.nay, accelgyro.naz) * 180) / 3.14;

  // Yaw Angle based on acelerometer
  //angleZ = accelGyroRelation * (angleZ + accelgyro.ngz * (float(dT) /
   ↪  1000000)) + (1 - accelGyroRelation) *
  //         (atan2(sqrt(pow(accelgyro.nax, 2) + pow(accelgyro.nay, 2)),
   ↪  accelgyro.naz) * 180) / 3.14;

  //  mxCalibrated = my - ((myMax + myMin) / 2.0);
  //  myCalibrated = mx - ((mxMax + mxMin) / 2.0);
  //  mzCalibrated = mz - ((mzMax + mzMin) / 2.0);
  //
  //  angleZ = atan2( (mzCalibrated * sin(angleX) - myCalibrated *
   ↪  cos(angleX)),
  //                  (mxCalibrated * cos(angleY) + myCalibrated *
   ↪  sin(angleY) * sin(angleX) + mzCalibrated * sin(angleY) *
   ↪  cos(angleX))  ) * (180 / 3.14);

  // Yaw Angle Based on Compass
  compassCalculation();
  angleZ = mediaMovel(reads);
}
```

```
/* To calculate heading in degrees. 0 degree indicates North
   Calculate in radians
   xC = mx - ((mxMax + mxMin) / 2.0);
   yC = my - ((myMax + myMin) / 2.0);
   zC = mz - ((mzMax + mzMin) / 2.0);
*/
void compassCalculation() { //256us
  //calculation
  compassAngle = atan2(my - ((myMax + myMin) / 2.0), mx - ((mxMax +
   ↪  mxMin) / 2.0)) * (180 / PI);
  compassAngle = compassAngle + 180;

  //check if it north
  if (compassAngle >= 360) {
    compassAngle -= 360;
  }
  if (compassAngle < 0) {
    compassAngle += 360;
  }

  //pass it for an array to stabilizate the data
  for (int i = 0; i < mediaMovelArray - 1; i++) {
    reads[i] = reads[i + 1];
  }
  reads[mediaMovelArray - 1] = compassAngle;

}

/*
   Calculate the media of an array
*/
float mediaMovel(float *vetor) {
  float soma = 0;
  for (int i = 0; i < mediaMovelArray; i++) {
    soma += vetor[i];
  }
  return soma / mediaMovelArray;
}
```

```
void compassCalibration() {
  mxMax = 0;
  mxMin = 0;
  myMax = 0;
  myMin = 0;
  mzMax = 0;
  mzMin = 0;

  long unsigned int t_cal = millis();
  while ((millis() - t_cal) < (CALIBRATING_TIME * 1000)) {
    wdt_reset();
    if (bluetoothSerial.available() > 0) {
      bluetoothSerial.read(); //flush buffer11
      bluetoothSerial.print("c,c\n");
    }
    mag.getHeading(&mx, &my, &mz);
    if (mxMax < mx) {
      mxMax = mx;
    }
    if (mxMin > mx) {
      mxMin = mx;
    }
    if (myMax < my) {
      myMax = my;
    }
    if (myMin > my) {
      myMin = my;
    }
    if (mzMax < mz) {
      mzMax = mz;
    }
    if (mzMin > mz) {
      mzMin = mz;
    }

#ifdef DEBUG
    Serial.print("Calibrating! Time remaining (s):");
```

```
    Serial.println(CALIBRATING_TIME - (millis() - t_cal) / 1000);
#endif
    delay(10);
  }

  EEPROM.write(0, mxMin >> 8);  EEPROM.write(1, mxMin % 256);
  EEPROM.write(2, myMin >> 8);  EEPROM.write(3, myMin % 256);
  EEPROM.write(4, mzMin >> 8);  EEPROM.write(5, mzMin % 256);
  EEPROM.write(6, mxMax >> 8);  EEPROM.write(7, mxMax % 256);
  EEPROM.write(8, myMax >> 8);  EEPROM.write(9, myMax % 256);
  EEPROM.write(10, mzMax >> 8);  EEPROM.write(11, mzMax % 256);

#ifdef DEBUG
  printCalibrationInfo();
#endif
}

void ressetCalibration() {
  mxMax = 0;
  mxMin = 0;
  myMax = 0;
  myMin = 0;
  mzMax = 0;
  mzMin = 0;
  EEPROM.write(0, mxMin >> 8);  EEPROM.write(1, mxMin % 256);
  EEPROM.write(2, myMin >> 8);  EEPROM.write(3, myMin % 256);
  EEPROM.write(4, mzMin >> 8);  EEPROM.write(5, mzMin % 256);
  EEPROM.write(6, mxMax >> 8);  EEPROM.write(7, mxMax % 256);
  EEPROM.write(8, myMax >> 8);  EEPROM.write(9, myMax % 256);
  EEPROM.write(10, mzMax >> 8);  EEPROM.write(11, mzMax % 256);
}

void initializeDevicesI2c() {

#ifdef DEBUG
  Serial.println("Initializing I2C devices...");
#endif


  wdt_reset();
```

```
  if (accelgyro.begin(MPU6050_SCALE_2000DPS, MPU6050_RANGE_2G,
   ↪  MPU6050_NORMAL, WIRE_400kHz)) {
#ifdef DEBUG
    Serial.print("\t MPU6050 connection successful \n");
#endif
    delay(10);
    accelgyro.setI2CMasterModeEnabled(false);
    accelgyro.setI2CBypassEnabled(true);
    accelgyro.setSleepEnabled(false);
    //accelgyro.setDHPFMode(MPU6050_DHPF_5HZ);
    accelgyro.setDLPFMode(MPU6050_DLPF_6);


#ifdef DEBUG
    Serial.print("\t MPU6050 configurate successful \n");
#endif

  } else {
#ifdef DEBUG
    Serial.print("\t *MPU6050 connection failed* \n");
#endif
  }

  mag.initialize();
#ifdef DEBUG
  Serial.print(mag.testConnection() ? "\t HMC5883L connection
   ↪  successful\n" : "*HMC5883L connection failed*\n");
#endif
}

/*
   Check I2C Lines, start bus and get device list
*/
void beginI2cBus() {

  wdt_reset();

  int rtn = I2C_ClearBus(); // clear the I2C bus first before calling
   ↪  Wire.begin()
```

```
  if (rtn != 0) {
#ifdef DEBUG
    Serial.print("\t I2C bus error. Could not clear \n");
#endif
    if (rtn == 1) {
#ifdef DEBUG
      Serial.print("\t SCL clock line held low \n");
#endif
    } else if (rtn == 2) {
#ifdef DEBUG
      Serial.print("\t SCL clock line held low by slave clock stretch
       ↪  \n");
#endif
    } else if (rtn == 3) {
#ifdef DEBUG
      Serial.print("\t SDA data line held low \n");
#endif
    }
  } else {
    // bus clear
    // re-enable Wire
    // now can start Wire Arduino master
    Wire.begin();
#ifdef DEBUG
    Serial.print("\t I2C bus clear. Init Sucesful\n");
    scanI2C();
#endif
  }
}


/*
   Clean I2C bus and return actual situation for start wire sucessfully
*/
int I2C_ClearBus() {

  wdt_reset();

#if defined(TWCR) && defined(TWEN)
```

```
  TWCR &= ~(_BV(TWEN)); //Disable the Atmel 2-Wire interface so we can
   ↪  control the SDA and SCL pins directly
#endif

  pinMode(SDA, INPUT_PULLUP); // Make SDA (data) and SCL (clock) pins
   ↪  Inputs with pullup.
  pinMode(SCL, INPUT_PULLUP);

  unsigned long timerI2c = millis();
  while (millis() - timerI2c < 2500) {
    wdt_reset(); // Wait 2.5 secs. This is strictly only necessary on the
     ↪  first power
  }
  // up of the DS3231 module to allow it to initialize properly,
  // but is also assists in reliable programming of FioV3 boards as it
   ↪  gives the
  // IDE a chance to start uploaded the program
  // before existing sketch confuses the IDE by sending Serial data.

  boolean SCL_LOW = (digitalRead(SCL) == LOW); // Check is SCL is Low.
  if (SCL_LOW) { //If it is held low Arduno cannot become the I2C master.
    return 1; //I2C bus error. Could not clear SCL clock line held low
  }

  boolean SDA_LOW = (digitalRead(SDA) == LOW);  // vi. Check SDA input.
  int clockCount = 20; // > 2x9 clock

  while (SDA_LOW && (clockCount > 0)) { //  vii. If SDA is Low,


    wdt_reset();

    clockCount--;
    // Note: I2C bus is open collector so do NOT drive SCL or SDA high.
    pinMode(SCL, INPUT); // release SCL pullup so that when made output
     ↪  it will be LOW
    pinMode(SCL, OUTPUT); // then clock SCL Low
    delayMicroseconds(10); //  for >5uS
    pinMode(SCL, INPUT); // release SCL LOW
```

```
  pinMode(SCL, INPUT_PULLUP); // turn on pullup resistors again
  // do not force high as slave may be holding it low for clock
   ↪  stretching.
  delayMicroseconds(10); //  for >5uS
  // The >5uS is so that even the slowest I2C devices are handled.
  SCL_LOW = (digitalRead(SCL) == LOW); // Check if SCL is Low.
  int counter = 20;
  while (SCL_LOW && (counter > 0)) {  //  loop waiting for SCL to
   ↪  become High only wait 2sec.

    wdt_reset();

    counter--;
    delay(100);
    SCL_LOW = (digitalRead(SCL) == LOW);
  }
  if (SCL_LOW) { // still low after 2 sec error
    return 2;
  }
  SDA_LOW = (digitalRead(SDA) == LOW); //   and check SDA input again
   ↪  and loop
}
if (SDA_LOW) { // still low
  return 3; // I2C bus error. Could not clear. SDA data line held low
}

// else pull SDA line low for Start or Repeated Start
pinMode(SDA, INPUT); // remove pullup.
pinMode(SDA, OUTPUT);  // and then make it LOW i.e. send an I2C Start
 ↪  or Repeated start control.
// When there is only one I2C master a Start or Repeat Start has the
 ↪  same function as a Stop and clears the bus.
/// A Repeat Start is a Start occurring after a Start with no
 ↪  intervening Stop.
delayMicroseconds(10); // wait >5uS
pinMode(SDA, INPUT); // remove output low
pinMode(SDA, INPUT_PULLUP); // and make SDA high i.e. send I2C STOP
 ↪  control.
delayMicroseconds(10); // x. wait >5uS
```

```
  pinMode(SDA, INPUT); // and reset pins as tri-state inputs which is the
   ↪  default state on reset
  pinMode(SCL, INPUT);
  return 0; // all ok
}

/*
   Scan i2c bus with a for() loking for adress in all 127 spaces.
*/
void scanI2C() {
  byte error, address;
  int nDevices;

  Serial.print("Scanning I2C Bus...\n");

  nDevices = 0;
  for (address = 1; address < 127; address++ ) {

    wdt_reset();

    // The i2c_scanner uses the return value of
    // the Write.endTransmisstion to see if
    // a device did acknowledge to the address.
    Wire.beginTransmission(address);
    error = Wire.endTransmission();

    if (error == 0) {
      Serial.print("\t I2C device found at address 0x");
      if (address < 16)
        Serial.print("0");
      Serial.println(address, HEX);

      nDevices++;
    }
    else if (error == 4) {
      Serial.print("\t Unknown error at address 0x");
      if (address < 16)
        Serial.print("0");
      Serial.println(address, HEX);
```

```
    }
  }
  if (nDevices == 0) {
    Serial.print("\t No I2C devices found\n");
  } else {
    Serial.print("\t Done scanner I2C\n");
  }
}

#include "configuration.h"

// Librarys includes
#include "Wire.h"
#include "MPU6050_bdt.h"
#include "HMC5883L_bdt.h"
#include "stp.h"
//#include <Stepper.h>

#include <SoftwareSerial.h>
#include <EEPROM.h>
#include <avr/wdt.h>

// Bluetooth variables
SoftwareSerial bluetoothSerial(16, 15); // RX/TX
unsigned long timerBluetoothRestartMessages = 0;
unsigned long timerBluetoothRecieveMessages = 0;
unsigned long timerBluetoothSendMessages = 0;
const byte bluetoothStatePin = 13;

//User Input Variables
char availableUserOption = '0';
bool manualDeactivation = true, remoteDeactivation = false;
bool manualActivation = false, lastManualActivation = false;
bool remoteActivation = false;

// Acelerometer variables
MPU6050 accelgyro;
float angleX, angleY, angleZ;
unsigned long gyroIntegrateTimer = 0; //Timer to interate Gyroscope axis
 ↪  and get Gyro part of Pitch and Roll angle.
```

```
const float accelGyroRelation = 0.5; //How mucth Gyro and Accel is
 ↪  consider for interate Pitch and Roll. 0 means that only accel will be
 ↪  used.

// Magnetometer variables
HMC5883L mag;
int mx, my, mz;
float mxCalibrated, myCalibrated, mzCalibrated;

// Calibration factors
int mxMin = EEPROM.read(0) << 8 | EEPROM.read(1);
int myMin = EEPROM.read(2) << 8 | EEPROM.read(3);
int mzMin = EEPROM.read(4) << 8 | EEPROM.read(5);
int mxMax = EEPROM.read(6) << 8 | EEPROM.read(7);
int myMax = EEPROM.read(8) << 8 | EEPROM.read(9);
int mzMax = EEPROM.read(10) << 8 | EEPROM.read(11);

// Calculation variables
const byte mediaMovelArray = 8;
float compassAngle = 0;
float reads[mediaMovelArray];

// Timer variables
unsigned long timerMpuData, timerStepperMicros = 0, timerLoop = 0;

// Stepper variables
boolean stepperState = false;
const int startButtonPin = 14;
const int stopButtonPin = 2;
const int stopButtonPin_sec = 3;
const int stepsPerRevolution = 4096 / 2;
HalfStepper myStepper = HalfStepper(stepsPerRevolution, 12, 10, 11, 9);

void setup() {
  //Initializate Whatchdog
  wdt_enable(WDTO_8S);

  // Initializate Variables
  for (int i = 0; i < mediaMovelArray; i++) {
```

```
    reads[i] = 0;
  }

  // Initializate Serial
#ifdef DEBUG
  Serial.begin(115200);
  printCalibrationInfo();
#endif

  // Initializate Bluetooth
  bluetoothSart();

  // Initializate I2C Bus
  beginI2cBus();
  initializeDevicesI2c();

  // Initializate Arduino pins
  //  pinMode(ledPin, OUTPUT);
  pinMode(startButtonPin, INPUT_PULLUP);
  pinMode(stopButtonPin, INPUT_PULLUP);
  pinMode(stopButtonPin_sec, INPUT_PULLUP);

  // Setup Stepper
  myStepper.setSpeed(STP_SPEED);
}

void loop() {

  wdt_reset();

  stepperState = checkStepperCondition();
  if (stepperState == true) {
    myStepper.step(runningSteps); //runs Stepper motor and elevate the
     ↪  plataform
    if (bluetoothSerial.available() > 0) {
      if (manualActivation && !remoteActivation) {
        bluetoothSerial.print("s,s\n");
      }
      availableUserOption = bluetoothSerial.read(); //flush buffer
```

```
    }
  } else {
    if (bluetoothSerial.available() > 0) {
      if (remoteActivation) {
        bluetoothSerial.print("n,n\n");
        remoteActivation = false;
      }
    }
  }

  if (stepperState == false) {
    getData();
    serialCommunication();
    bluetoothCommunication();
  }

  manageUserOption();

#ifdef DEBUG
//  Serial.print("manualActivation "); Serial.println(manualActivation);
//  Serial.print("manualDeactivation ");
 ↪  Serial.println(manualDeactivation);
//  Serial.print("remoteActivation "); Serial.println(remoteActivation);
//  Serial.print("remoteDeactivation ");
 ↪  Serial.println(remoteDeactivation); Serial.println();
#endif
}


bool checkStepperCondition() {
  int nullvar = analogRead(startButtonPin); //não tem função nenhuma, não
   ↪  serve pra nada, porém o código só funciona se estiver aqui!
  if (millis() > 5000) {// dont get the starting set of pullup as a true
   ↪  value
    if (!manualActivation && !digitalRead(startButtonPin)) {
      manualActivation = true;
      manualDeactivation = false;
      remoteDeactivation = false;
    }
```

```
    if (manualActivation && digitalRead(startButtonPin)) {
      manualActivation = false;
      manualDeactivation = true;
      remoteActivation = false;
    }
  }
  bool stopState = (!digitalRead(stopButtonPin) ||
   ↪  !digitalRead(stopButtonPin_sec));
  if(stopState){
    manualActivation = false;
    remoteActivation = false;
  }
  return ((manualActivation || remoteActivation) && (!manualDeactivation
   ↪  && !remoteDeactivation && !stopState));
}

bool checkBluetoothCondition() {
  return (!digitalRead(bluetoothStatePin));
}

void serialCommunication() {
#ifdef DEBUG
  while (Serial.available() > 0) {
    availableUserOption = Serial.read();
  }
#endif
}


/*
   @brief
   @note
   @param
   @return
*/
void manageUserOption() {
  if (availableUserOption == 'c') {
    compassCalibration();
  }
```

```
  if (availableUserOption == 's') {
    remoteActivation = true;
    remoteDeactivation = false;
    manualDeactivation = false;
    bluetoothSerial.print("s,s\n"); //while writebuffer isnt clean in
     ↪  android, send the buffer that trigger an internal clean
  }
  if (availableUserOption == 'n') {
    remoteDeactivation = true;
    remoteActivation = false;
    bluetoothSerial.print("n,n\n"); //while writebuffer isnt clean in
     ↪  android, send the buffer that trigger an internal clean
  }
  if (availableUserOption == 'r') {
    ressetCalibration();
  }
  if (availableUserOption == '1') {
    remoteActivation = true;
    remoteDeactivation = false;
  }
  availableUserOption = '0';
}
```

APÊNDICE D – CÓDIGO DO APLICATIVO (SEM AS TELAS)

D.1 – ROTINA DA ATIVIDADE PRINCIPAL

```
package com.example.startracker


import android.content.Context
import android.os.Bundle
import android.view.MotionEvent
import android.view.View
import android.view.inputmethod.InputMethodManager
import android.widget.EditText
import androidx.appcompat.app.AppCompatActivity
import androidx.appcompat.app.AppCompatDelegate
import androidx.appcompat.app.AppCompatDelegate.MODE_NIGHT_YES
import androidx.appcompat.widget.Toolbar
import androidx.drawerlayout.widget.DrawerLayout
import androidx.navigation.NavController
import androidx.navigation.NavDestination
import androidx.navigation.findNavController
import androidx.navigation.fragment.NavHostFragment
import androidx.navigation.ui.NavigationUI
import com.example.startracker.databinding.ActivityMainBinding


class MainActivity : AppCompatActivity() {

    var hc05 = BluetoothService()
    lateinit var toolbar: Toolbar
    private lateinit var drawerLayout: DrawerLayout
    private lateinit var binding: ActivityMainBinding
    private lateinit var destinationHandler: NavDestination

    override fun onCreate(savedInstanceState: Bundle?) {
        setTheme(R.style.Theme_MyApplication_NoActionBar)
        super.onCreate(savedInstanceState)
```

```
binding = ActivityMainBinding.inflate(layoutInflater)
setContentView(binding.root)

// set the Android Night Mode as default for the app
AppCompatDelegate.setDefaultNightMode(MODE_NIGHT_YES)

// get toolbar instance
toolbar = binding.toolbar
setSupportActionBar(toolbar)

drawerLayout = binding.drawerLayout

//Get and setup navcontroller that will make the navigation
 ↪  between fragments
val navHostFragment = supportFragmentManager
    .findFragmentById(R.id.nav_host_fragment) as NavHostFragment

val navController = navHostFragment.navController
val graphInflater = navHostFragment.navController.navInflater
val navGraph = graphInflater.inflate(R.navigation.nav_graph)

NavigationUI.setupActionBarWithNavController(this, navController,
 ↪  drawerLayout)
NavigationUI.setupWithNavController(binding.navView,
 ↪  navController)

//change home screen in agreement with the case of new user or
 ↪  not
//TODO: check if this function can be removed
navController.addOnDestinationChangedListener { nc:
 ↪  NavController,
                                                nd:
                                                 ↪  NavDestination,
                                                args: Bundle? ->
    if (nd.id == R.id.currentProfileFragment) {
        navGraph.startDestination = R.id.currentProfileFragment
        setDrawer_locked()
    }
```

```
    }

    // check if the screen is for new user, and remove/locking
    // navigation parts of the app, making it clean
    navController.addOnDestinationChangedListener { nc:
     ↪ NavController,
                                                    nd:
                                                     ↪ NavDestination,
                                                    args: Bundle? ->
        if (nd.id == R.id.welcomeFragment) {
            navGraph.startDestination = R.id.welcomeFragment
            setDrawer_locked()
            toolbar.navigationIcon = null
        }
    }

    //set the back button menu to be hide at home screen
    navController.addOnDestinationChangedListener { nc:
     ↪ NavController,
                                                    nd:
                                                     ↪ NavDestination,
                                                    args: Bundle? ->
        destinationHandler = nd
        if (nd.id == nc.graph.startDestination) {
            //setDrawer_locked()
            setDrawer_unLocked()
        } else {
            //setDrawer_unLocked()
            setDrawer_locked()
        }
    }

}

//activate the navigation in acivity
override fun onSupportNavigateUp(): Boolean {
    val navController =
     ↪ this.findNavController(R.id.nav_host_fragment)
```

```
    return NavigationUI.navigateUp(navController, drawerLayout)
}

//set the back button menu to be displayed or not
fun setDrawer_locked() {

    ↪ drawerLayout.setDrawerLockMode(DrawerLayout.LOCK_MODE_LOCKED_CLOSED)
    //toolbar.setNavigationIcon(null)
}

fun setDrawer_unLocked() {
    drawerLayout.setDrawerLockMode(DrawerLayout.LOCK_MODE_UNLOCKED)
}

//block native back button
override fun onBackPressed() {
    if (shouldAllowBack()) {
        super.onBackPressed()
    }
}

//block native back button for ensure that the new user wont back to
// the welcome screen manualy and/or replicate the same profile
private fun shouldAllowBack(): Boolean {
    if (destinationHandler.id == R.id.currentProfileFragment) {
        return false
    }

    return true
}

//Hide Keyboard when user touch outside
//https://stackoverflow.com/questions/8697499/
override fun dispatchTouchEvent(ev: MotionEvent): Boolean {
    val view: View? = currentFocus
    if (view != null && (ev.action == MotionEvent.ACTION_UP ||
                ev.action == MotionEvent.ACTION_MOVE) && view is
                 ↪ EditText
    ) {
```

```
            val scrcoords = IntArray(2)
            view.getLocationOnScreen(scrcoords)
            val x: Float = ev.rawX + view.getLeft() - scrcoords[0]
            val y: Float = ev.rawY + view.getTop() - scrcoords[1]
            if (x < view.getLeft() || x > view.getRight() || y <
             ↪  view.getTop()
                || y > view.getBottom()
            ) (this.getSystemService(
                Context.INPUT_METHOD_SERVICE
            ) as InputMethodManager).hideSoftInputFromWindow(
                this.window.decorView.applicationWindowToken, 0
            )
        }
        return super.dispatchTouchEvent(ev)
    }
}
```

```
package com.example.startracker

import android.annotation.SuppressLint
import android.content.Context
import android.util.DisplayMetrics
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import java.text.SimpleDateFormat

// File with functions used in all code

//sensor error margin for user get screen green
var errorMargin = 0.5

/**
 * This method converts dp unit to equivalent pixels, depending on device
 ↪  density.
 *
 * @param dp A value in dp (density independent pixels) unit. Which we
 ↪  need to convert into pixels
 * @param context Context to get resources and device specific display
 ↪  metrics
```

```
 * @return A float value to represent px equivalent to dp depending on
 ↪  device density
 */
fun convertDpToPixel(dp: Float, context: Context): Float {
    return dp * (context.getResources()
        .getDisplayMetrics().densityDpi.toFloat() /
         ↪  DisplayMetrics.DENSITY_DEFAULT)
}

/**
 * This method converts device specific pixels to density independent
 ↪  pixels.
 *
 * @param px A value in px (pixels) unit. Which we need to convert into
 ↪  db
 * @param context Context to get resources and device specific display
 ↪  metrics
 * @return A float value to represent dp equivalent to px value
 */
fun convertPixelsToDp(px: Float, context: Context): Float {
    return px / (context.getResources()
        .getDisplayMetrics().densityDpi.toFloat() /
         ↪  DisplayMetrics.DENSITY_DEFAULT)
}

/**
 * Take the Long milliseconds returned by the system and stored in Room,
 * and convert it to a nicely formatted string for display.
 *
 * EEEE - Display the long letter version of the weekday
 * MMM - Display the letter abbreviation of the nmotny
 * dd-yyyy - day in month and full year numerically
 * HH:mm - Hours and minutes in 24hr format
 */
@SuppressLint("SimpleDateFormat")
fun convertLongToDateString(systemTime: Long, patern:String): String {
    return SimpleDateFormat(patern)
        .format(systemTime).toString()
}
```

```
fun mapFloat(keyValue: Float, in_min: Float, in_max: Float, out_min:
 ↪  Float, out_max: Float):Float{
    return (keyValue - in_min) * (out_max - out_min) / (in_max - in_min)
     ↪  + out_min
}

/**
 * ViewHolder that holds a single [TextView].
 *
 * A ViewHolder holds a view for the [RecyclerView] as well as providing
 ↪  additional information
 * to the RecyclerView such as where on the screen it was last drawn
 ↪  during scrolling.
 */
class TextItemViewHolder(val textView: TextView):
 ↪  RecyclerView.ViewHolder(textView)
```

D.2 – LAYOUT DA ATIVIDADE PRINCIPAL

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.coordinatorlayout.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

    <com.google.android.material.appbar.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

```
        android:theme="@style/Theme.MyApplication.AppBarOverlay">

        <androidx.appcompat.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            app:popupTheme="@style/Theme.MyApplication.PopupOverlay"
            />

    </com.google.android.material.appbar.AppBarLayout>

        <androidx.drawerlayout.widget.DrawerLayout
            android:id="@+id/drawerLayout"
            android:layout_width="match_parent"
            android:layout_height="match_parent">

            <androidx.fragment.app.FragmentContainerView
                android:id="@+id/nav_host_fragment"

                 ↪  android:name="androidx.navigation.fragment.NavHostFragment"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                app:defaultNavHost="true"
                app:layout_anchor="@+id/drawerLayout"
                app:layout_anchorGravity="center"
                app:navGraph="@navigation/nav_graph" />

            <com.google.android.material.navigation.NavigationView
                android:id="@+id/navView"
                android:layout_width="wrap_content"
                android:layout_height="match_parent"
                android:layout_gravity = "start"
                app:menu="@menu/nav_menu"
                app:headerLayout="@layout/nav_header"/>

        </androidx.drawerlayout.widget.DrawerLayout>
    </LinearLayout>
</androidx.coordinatorlayout.widget.CoordinatorLayout>
```

D.3 – CLASSE BLUETOOTH

```
package com.example.startracker

import android.bluetooth.BluetoothAdapter
import android.bluetooth.BluetoothDevice
import android.bluetooth.BluetoothSocket
import android.os.Bundle
import android.os.Handler
import android.os.Looper
import android.os.Message
import android.util.Log
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import java.io.IOException
import java.io.InputStream
import java.io.OutputStream
import java.util.*
import kotlin.concurrent.thread

class BluetoothService {

    // Defines several constants used when transmitting messages between
     ↪  the
    // service and the UI.
    val MESSAGE_READ: Int = 0

    private var _mmIsConnected = MutableLiveData<Boolean?>()
    val mmIsConnected: LiveData<Boolean?>
        get() = _mmIsConnected

    private var _calibratingCompass = MutableLiveData<Boolean>()
    val calibratingCompass: LiveData<Boolean>
        get() = _calibratingCompass

    var trackingStars = MutableLiveData<Boolean>().apply { value =
     ↪  (false)}

    // raw data get in bluetooth connection
```

```
private var _rawDataRoll: Int = 0
private var _rawDataPitch: Int = 0
private var _rawDataYaw: Int = 0
private var _rawDataCRC: Int = 0
private var _rawDataError: Boolean = false

// data converted from bluetooth = raw/10
private var _dataRoll = MutableLiveData<Float>()
val dataRoll: LiveData<Float>
    get() = _dataRoll

private var _dataPitch = MutableLiveData<Float>()
val dataPitch: LiveData<Float>
    get() = _dataPitch

private var _dataYaw = MutableLiveData<Float>()
val dataYaw: LiveData<Float>
    get() = _dataYaw

private var _dataError1 = MutableLiveData<Boolean>()
val dataError1: LiveData<Boolean>
    get() = _dataError1

private var _updatedHandle = MutableLiveData<Boolean>()
val updatedHandle: LiveData<Boolean>
    get() = _updatedHandle

// buffer read in bluetooth
var stringBuffer: String = "0,0,0,0"

// bluetooth address of device
private var mmDeviceMAC: String = ""

// thread that read bluetooth buffer data
lateinit var RunnableThread: ConnectedThread

init {
    _mmIsConnected.value = null
    _updatedHandle.value = false
```

```
    _calibratingCompass.value = false
}

// The Handler that gets information back from the BluetoothService
private val handler = object : Handler(Looper.getMainLooper()) {
    override fun handleMessage(msg: Message) {
        val bundle: Bundle = msg.data

        // get buffer String
        // println(stringBuffer)
        stringBuffer = bundle.getString("key1", stringBuffer)
        val dataString = stringBuffer.split(",").toTypedArray()

        // check if buffer array have only 4 values
        if (dataString.size == 4) {
            try {
                _rawDataRoll = dataString[0].toInt()
                _rawDataPitch = dataString[1].toInt()
                _rawDataYaw = dataString[2].toInt()
                _rawDataCRC = dataString[3].toInt()
                //_rawDataError:Boolean = dataString[0].toInt()
            } catch (e: Exception) {
                Log.e("DEBUGCONNECTION", "Data Values with error", e)
            }
            // launch another thread for update UI values IF this
            ↪  values match CRC
            // CRC is just a sum with the other values
            thread {
                if ((_rawDataRoll + _rawDataPitch + _rawDataYaw) ==
                ↪  _rawDataCRC) {
                    //updateWriteBuffer("0")
                    _dataRoll.postValue(_rawDataRoll.toFloat() / 10)
                    _dataPitch.postValue(_rawDataPitch.toFloat() /
                    ↪  10)
                    _dataYaw.postValue(_rawDataYaw.toFloat() / 10)
                    _updatedHandle.postValue(!_updatedHandle.value!!)
                    _calibratingCompass.postValue(false)
                }
            }
```

```
        }else if(dataString.size == 2){
            thread {
                if (dataString[0] == dataString[1]) {
                    if(dataString[0] == "s"){
                        trackingStars.postValue(true)
                        updateWriteBuffer("1")
                    }else if(dataString[0] == "n"){
                        trackingStars.postValue(false)
                        updateWriteBuffer("2")
                    }else if (dataString[0] == "c") {
                        _calibratingCompass.postValue(true)
                        updateWriteBuffer("3")
                    }
                }
            }
        }
    }

    /**
     * Connect with a bluetooth device specified by your address
     * Launch this in a separated thread
     * @param DeviceMAC that is the bluetooth address
     */
    fun connect(DeviceMAC: String) {
        thread {
            try {
                if (_mmIsConnected.value != true) {
                    mmDeviceMAC = DeviceMAC
                    RunnableThread = ConnectedThread(DeviceMAC, handler)
                    RunnableThread.connectThread()

                     ↪  _mmIsConnected.postValue(RunnableThread.mmThreadIsConnected
                    if (RunnableThread.mmThreadIsConnected) {
                        RunnableThread.run()
                    } else {
                        _mmIsConnected.postValue(false)
                    }
                }
```

```
        } catch (e: java.lang.Exception) {
            e.printStackTrace()
        }
    }
}

/**
 * Disconnect with the current bluetooth device
 * Launch this in a separated thread
 */
fun disconnect() {
    thread{
        try {
            if (_mmIsConnected.value == true) {
                RunnableThread.disconnectThread()

                ↪  _mmIsConnected.postValue(RunnableThread.mmThreadIsConnected
            }

             ↪  _mmIsConnected.postValue(RunnableThread.mmThreadIsConnected)
        } catch (e: java.lang.Exception) {
            e.printStackTrace()
        }
    }
}

/**
 * Reconnect with a bluetooth device specified by your address
 * Launch this in a separated thread
 */
fun reconnect() {
    thread{
        try {
            if (_mmIsConnected.value == true) { //disconect just if
             ↪  it is connected
                RunnableThread.disconnectThread()
            }
            RunnableThread = ConnectedThread(mmDeviceMAC, handler)
            RunnableThread.connectThread()
```

```
                ↪ _mmIsConnected.postValue(RunnableThread.mmThreadIsConnected)
                if (RunnableThread.mmThreadIsConnected) {
                    RunnableThread.run()
                } else {
                    _mmIsConnected.postValue(false)
                }
            } catch (e: java.lang.Exception) {
                e.printStackTrace()
            }
        }
    }

    fun updateWriteBuffer(buffer: String){
        RunnableThread.writeBuffer = buffer
    }

    //class that stands for run a thread for read and write in bluetooth
     ↪ device
    inner class ConnectedThread(var DeviceMAC: String, var handler:
     ↪ Handler) : Thread() {

        // bluetooth variables
        private lateinit var mmDevice: BluetoothDevice
        private lateinit var mmAdapter: BluetoothAdapter
        private lateinit var mmInStream: InputStream
        private lateinit var mmOutStream: OutputStream
        lateinit var mmSocket: BluetoothSocket
        var writeBuffer = "0"

        //state of connection
        var mmThreadIsConnected = false
        var mmThreadIsDesconnecting = false

        // Unique UUID for this application
        // https://stackoverflow.com/questions/32130529/
        val myUUID: UUID? =
         ↪ UUID.fromString("00001101-0000-1000-8000-00805F9B34FB")
```

```
// runs the thread for communicate with HC05
override fun run() {

    // check if it still connected
    if (mmThreadIsConnected == true) {

        var getWriteTime: Long = System.currentTimeMillis()
        val buffer = ByteArray(1)
        var bytes: Int
        var readMessage = ""
        var readChar: String

        var getTime: Long = System.currentTimeMillis()
        while (true) {
            // Keep listening to the InputStream until an
             ↪ exception occurs.
            try {
                //ensure that the buffer is allways clean
                //delaytime is the delay that ui wait for read
                 ↪ next buffer.
                var delaytime = 10
                if ((mmInStream.available() > 1000)) {
                    // if the buffer is overload, i.e, is bigger
                     ↪ then 230 bytes,
                    // reduce the delay so the UI can update
                     ↪ faster enough
                    delaytime = 1
                }
                //if available and also get a delay between reeds
                if ((mmInStream.available() > 0) &&
                    ((System.currentTimeMillis() - getTime) >
                     ↪ delaytime)
                ) {

                    bytes = mmInStream.read(buffer) //read bytes
                     ↪ from input buffer
                    readChar = String(buffer, 0, bytes) //get
                     ↪ Char
```

```
            //if char isnt '\n' then join all char
             ↪  recieved,
            // mounting the data string. Stop when '\n'
             ↪  and send it through handler
            if (readChar == "\n") {
                getTime = System.currentTimeMillis()
                val readMsg =
                 ↪  handler.obtainMessage(MESSAGE_READ)
                val bundle = Bundle()
                bundle.putString("key1", readMessage)
                readMsg.data = bundle
                readMsg.sendToTarget()
                readMessage = ""
            } else {
                readMessage += readChar
            }
        }
    } catch (e: IOException) {
        Log.e("DEBUGCONNECTION", "Input stream was
         ↪  disconnected")
        // in case of error, start disconnect if it not
         ↪  started
        if (!mmThreadIsDesconnecting) {
            mmThreadIsDesconnecting = true
            disconnect()
        }
        break
    }

    // launch another thread to write in output buffer
    thread {
        // send messages with 2Hz of speed
        if ((System.currentTimeMillis() - getWriteTime) >
         ↪  500) {
            getWriteTime = System.currentTimeMillis()
            write(writeBuffer.encodeToByteArray())
        }
    }
```

```
        }
    }
}

// Call this from the main activity to send data to the remote
 ↪  device.
fun write(bytes: ByteArray) {
    try {
        mmOutStream.write(bytes)
    } catch (e: IOException) {
        Log.e("DEBUGCONNECTION", "Error occurred when sending
         ↪  data")

        if (!mmThreadIsDesconnecting) {
            mmThreadIsDesconnecting = true
            disconnect()
        }

        return
    } catch (e: Exception) {
        Log.e("DEBUGCONNECTION", "Error occurred when sending
         ↪  data")
    }
}

// This function check if the phone has bt adapter
// and then connect with device using the UUID
fun connectThread() {
    if (BluetoothAdapter.getDefaultAdapter() != null) {

        mmAdapter = BluetoothAdapter.getDefaultAdapter()
        mmThreadIsDesconnecting = false

        if (!mmAdapter.isEnabled) {
            throw Exception("Bluetooth adapter not found or not
             ↪  enabled!")
        }

        mmDevice = mmAdapter.getRemoteDevice(DeviceMAC)
```

```
            mmSocket =
             ↪ mmDevice.createRfcommSocketToServiceRecord(myUUID)

            try {
                mmSocket.connect()
                mmThreadIsConnected = mmSocket.isConnected
                // Cancel discovery because it otherwise slows down
                 ↪ the connection.
                mmAdapter.cancelDiscovery()
                mmInStream = mmSocket.inputStream
                mmOutStream = mmSocket.outputStream

            } catch (e: IOException) {
                mmThreadIsConnected = mmSocket.isConnected
                Log.e("DEBUGCONNECTION", "UNABLE TO CONNECT WITH
                 ↪ BLUETOOTH DEVICE")
            }
        } else {
            Log.e("DEBUGBLUETOOTH", "DONT HAVE BLUETOOTH ADAPTER")
        }
    }

    // disconnect stream, socket and bluetooth device
    fun disconnectThread() {
        try { // close input
            mmInStream.close()
        } catch (e: IOException) {
            e.printStackTrace()
        }
        try { // close output
            mmOutStream.close()
        } catch (e: IOException) {
            e.printStackTrace()
        }
        try { // close socket
            mmSocket.close()
        } catch (e: IOException) {
            e.printStackTrace()
```

```
                Log.e("DEBUGCONNECTION", "Could not close the connect
                    ↪ socket")
            } catch (e: Exception) {
                e.printStackTrace()
                Log.e("DEBUGCONNECTION", "Could not close the connect
                    ↪ socket")
            }
            // update state
            mmThreadIsConnected = mmSocket.isConnected
        }
    }
}
```