UNIVERSIDADE DO MINHO
Janeiro 2018

Mestrado Integrado em Engenharia Eletrónica Industrial e Computadores

UNIDADE CURRICULAR
Projeto I

Pêndulo Invertido

Robô Equilibrista – “SpongeBob”

Docentes: Filomena Soares e João Sena Esteves

Alunos:

José Pedro Araújo Soares- a75127

Luís Miguel Matos de Almeida- a68576

Índice

Conteúdo

Índice de figuras

Índice de tabelas

Sumário

Neste trabalho propôs-se construir um robô equilibrista de raiz, para isso utilizou-se madeira para a construção do robô.

O objetivo principal deste projeto seria desenvolver um robô capaz de se autoequilibrar, tendo outros objetivos como criação de uma aplicação android e a sua navegação autónoma.

Assim, criou-se uma aplicação, no MIT APP INVENTOR, para fornecer a direção desejada pelo utilizador ao robô. Utilizou-se o PID de velocidade para o controlo, cumprindo assim o objetivo de o robô equilibrar-se.

Por fim, para concluir o ultimo objetivo seria necessário a utilização dos encoders. Existindo alguns problemas, um dos encoders estava avariado, foi possível concluir esse objetivo, tendo o grupo maiores dificuldades nessa etapa.

Introdução

Motivação e enquadramento

No âmbito da disciplina de Projeto I fez-se a escolha deste projeto cujo objetivo é a criação física de um pendulo invertido.

Após o tema escolhido, pesquisou-se e selecionou-se uma estrutura apropriada para o robô.

Inicialmente, pensou-se num robô que se equilibra com uma roda, mas devido aos problemas mecânicos que viríamos a ter, especialmente a mudança de direção, mudou-se para um robô com duas rodas.

Objetivos e estudo do problema

O principal objetivo deste projeto é desenvolver um controlo que permita equilibrar um robô que possui o comportamento de um pendulo invertido, podendo-se dividir este projeto em seis objetivos, no primeiro objetivo, estudou-se o problema. Observou-se qual a melhor forma de construir o robô e qual o controlo que se iria utilizar. No segundo objetivo, avançou-se para a montagem do robô. No terceiro objetivo, testou-se todos os componentes que o robô iria utilizar (motores, acelerómetro…). No quarto objetivo, usou-se o controlo PID de velocidade e aplicou-se no robô. No quinto objetivo, fez-se uma aplicação para controlar a direção do robô. Por fim, no sexto objetivo, testou-se os encoders e aplicou-se a navegação autónoma.

Para equilibrar-se o robô é necessário conhecer o seu ângulo de inclinação (variável a controlar), de modo a transmitir aos motores a potência necessária para manter o robô na nossa referência. Com os encoders das rodas calculou-se a velocidade e a posição do robô. Por fim aplicou-se um PID de modo a reagir ás perturbações que o sistema sofre.

Estado da Arte

Atualmente existem vários projetos baseados no controlo de um pêndulo invertido, como por exemplo o *Segway*, que foi criado por Dean Kamen, ou brinquedos como o *MiP*, que foi criado por Wowwee.

Figura 1 - Dean Kamen no Segway

O Segway, figura 1, foi apresentado em Dezembro de 2001 e é fabricado pela empresa *Segway Inc*. Este veiculo caracterizasse por ser elétrico, tendo uma autonomia de 38km e uma velocidade máxima de 20km/h, modelo i2.

O *Segway* movimenta-se consoante a inclinação imposta por uma pessoa que está em cima do mesmo, quando se inclina para a frente o *Segway* anda para a frente e vice-versa.

Figura 2 - MiP, o robô equilibrista

O Mip é um robô que anda sobre duas rodas, construído pela Wowwee podendo ser controlado não só por um telemóvel ou tablet mas sim com os movimentos da mão.

Vantagens dos robôs equilibristas sobre duas rodas

1- Conseguem rodar sobre si mesmos.
2- O seu centro de gravidade é situado por cima das rodas em todos os momentos.

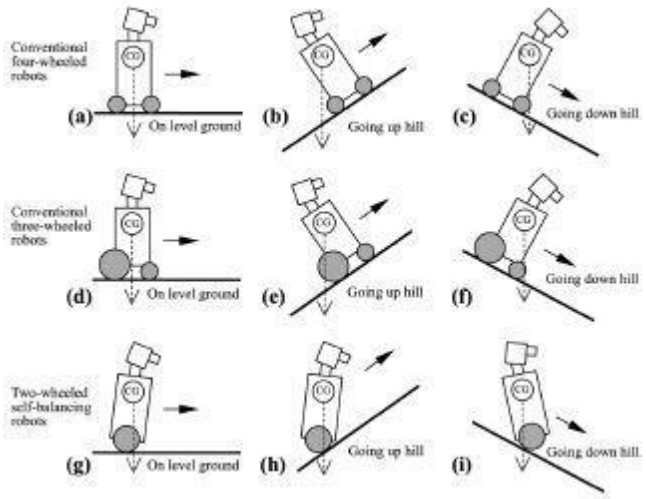


Figura 3 - Centro de gravidade

Divisão do Relatório

O relatório tem a seguinte estrutura:

1- Na primeira parte apresentou-se os principais fundamentos teóricos.

2- Na segunda parte descreveu-se ao trabalho realizado pelo grupo.

3- Na terceira parte apresentou-se os testes realizados.

4- Por ultimo, desenvolveu-se a conclusão, bibliografia e anexos utilizados.

Fundamentos Teóricos

O robô equilibrista é por natureza um sistema instável, sendo que o centro de gravidade é situado na parte de baixo do robô e centralizado entre as duas rodas. Devido a estas condições, é desejado um sistema de controlo que detete o grau e a velocidade da inclinação do robô, corrigindo a sua posição aplicando a potência necessária aos motores para manter o robô na nossa referência.

Análise matemática do modelo

O modelo matemático do robô equilibrista é um modelo complexo, utilizou-se o artigo em anexo A.1 para se entender o funcionamento e assim, de seguida apresentou-se o modelo matemático do robô.

Os parâmetros do modelo são apresentados na tabela abaixo, *tabela 1*.

Tabela 1 - Parâmetros do sistema

Parâmetro	Unidade	Descrição
θ	[graus]	Ângulo das rodas
ψ	[graus]	Ângulo de inclinação do corpo
Φ	[graus]	Ângulo de desvio do corpo
θm	[graus]	Ângulo do motor
g	[m/seg^2]	Constante de aceleração gravítica
m	[kg]	Peso da roda
R	[m]	Raio da roda
Jw	[kgm^2]	Momento de inercia da roda
M	[kg]	Peso do corpo
W	[m]	Largura do corpo
D	[m]	Profundidade do corpo
H	[m]	Altura do corpo
L	[m]	Distância do centro de massa ao eixo das rodas
J	[kgm^2]	Momento de inercia
J_ψ	[kgm^2]	Momento de inercia de desvio do corpo
$J\Phi$	[kgm^2]	Momento de inercia de inclinação do

		corpo
J_m	[kgm^2]	Momento de inercia do motor
N		Relação de transmissão
K_t	[Nm/A]	Constante de torque do motor
Kb	[V seg./rad]	Constante de EMF do motor
f_m		Coeficiente de fricção entre o corpo e o motor
fw		Coeficiente de fricção entre a roda e o chão

Utilizou-se o modelo do sistema da *Figura 4,* para se obter as equações do movimento.

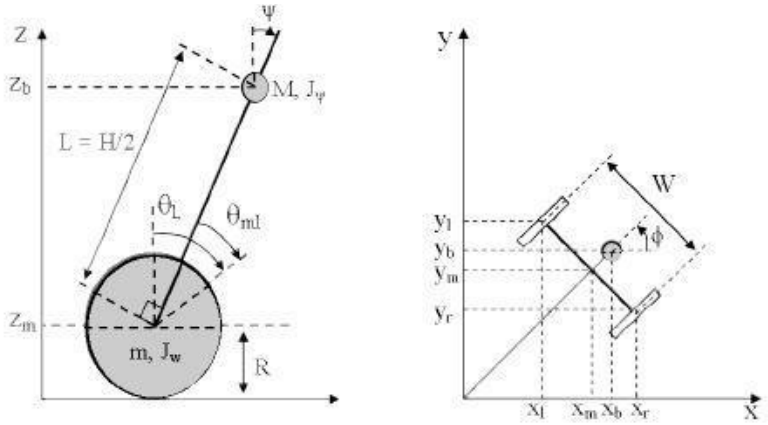


Figura 4 - Modelo do sistema

As equações de movimento linearizadas do sistema são descritas abaixo:

$$F_{\theta} = \left((2m + M)R^2 + 2J_{\Psi} + 2n^2J_m\right)\ddot{\Theta} + (MLR - 2n^2J_m)\ddot{\Psi} \quad \text{Equação 1}$$

$$F_{\Psi} = (MLR - 2n^2J_m)\ddot{\Theta} + (ML^2 + J_{\Psi} + 2n^2J_m)\ddot{\Psi} - MgL\Psi \quad \text{Equação 2}$$

$$F_{\Phi} = \left(\frac{1}{2}\, mW^2 + J_{\Phi} + \frac{w^2}{2R^2}\,(J_w + n^2J_m)\right)\ddot{\Phi} \quad \text{Equação 3}$$

A equação 1 e a equação 2 podem ser agrupadas, pois possuem os ângulos ɵ e Ψ. A equação 3 só tem o angulo de desvio, Φ. Assim, descreveu-se as duas primeiras equações:

$$A\begin{bmatrix}\ddot{\Theta}\\ \ddot{\Psi}\end{bmatrix} + B\begin{bmatrix}\dot{\theta}\\ \dot{\Psi}\end{bmatrix} + C\begin{bmatrix}\Theta\\ \Psi\end{bmatrix} = U\begin{bmatrix}V_L\\ V_R\end{bmatrix} \quad \text{Equação 4}$$

Onde,

$$A = \begin{bmatrix}(2m + M)R^2 + 2J_{\Psi} + 2n^2J_m & MLR - 2n^2J_m\\ MLR - 2n^2J_m & ML^2 + J_{\Psi} + 2n^2J_m\end{bmatrix}$$

$$B = 2\begin{bmatrix}\beta + f_W & -\beta\\ \beta & \beta\end{bmatrix}$$

$$C = 2\begin{bmatrix}0 & 0\\ 0 & -MgL\end{bmatrix}$$

$$U = 2\begin{bmatrix}\alpha & \alpha\\ -\alpha & -\alpha\end{bmatrix}$$

$$\alpha\frac{nK_t}{R_m}, \beta = \frac{nK_tK_b}{R_m} + f_m$$

A equação 3 pode ser descrita da seguinte forma:

$$I\ddot{\Phi} + J\ddot{\Phi} = K(V_RV_L) \quad \text{Equação 5}$$

Onde,

$$I = \frac{1}{2}\, mW^2 + J_{\Phi} + \frac{w^2}{2R^2}\,(J_w + n^2J_m)$$

$$J = \frac{w^2}{2R^2}(\beta + f_W)$$

$$K = \frac{W}{2R}a$$

Usando as variáveis x_1 e x_2 como variáveis de estado para as equações 4 e 5, e u como a entrada do sistema, as equações do espaço de estado tornam-se:

$$x_1 = \begin{bmatrix} \theta \\ \psi \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \quad x_2 = \begin{bmatrix} \Phi \\ \dot{\Phi} \end{bmatrix} \quad u = \begin{bmatrix} v_l \\ v_r \end{bmatrix} \qquad \text{Equação 6}$$

Assim,

$$\dot{x}_1 = A_1 x_1 + B_1 u \qquad \text{Equação 7}$$

$$\dot{x}_2 = A_2 x_2 + B_2 u \qquad \text{Equação 8}$$

$$A_1 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & A_1(3,2) & A_1(3,3) & A_1(3,4) \\ 0 & A_1(4,2) & A_1(4,3) & A_1(4,4) \end{bmatrix}, B_1 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ B_1(3) & B_1(3) \\ B_1(4) & B_1(4) \end{bmatrix}$$

$$A_2 = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{J}{I} \end{bmatrix}, B_2 = \begin{bmatrix} 0 & 0 \\ -\frac{K}{I} & \frac{K}{I} \end{bmatrix}$$

$$A_1(3,2) = -\frac{gMLE(1,2)}{\det(E)}$$

$$A_1(4,2) = \frac{gMLE(1,1)}{\det(E)}$$

$$A_1(3,3) = -\frac{2[\beta + f_W)E(2,2) + \beta E(1,2)]}{\det(E)}$$

$$A_1(4,3) = \frac{2[\beta + f_W)E(1,2) + \beta E(1,1)]}{\det(E)}$$

$$A_1(3,4) = \frac{2\beta[E(2,2) + E(1,2)]}{\det(E)}$$
$$A_1(4,4) = -\frac{2\beta[E(1,1) + E(1,2)]}{\det(E)}$$
$$B_1(3) = \frac{\alpha[E(2,2) + E(1,2)]}{\det(E)}$$
$$B_1(4) = -\frac{\alpha[E(1,1) + E(1,2)]}{\det(E)}$$
$$det(E) = E(1,1)E(2,2) - E(1,2)^2$$

Controlador PID

O PID(Proporcional-Integral-Derivativo) é o algoritmo de controlo mais usado na indústria, devido a sua flexibilidade, aplicação em diversas aplicações e obtenção de bons resultados.

A ação proporcional produz um sinal de saída que é proporcional à amplitude do erro e(t), sendo K_p a constante de proporcionalidade.

$$P_{saida} = K_p e^{(t)} \quad \text{Equação 9}$$

A ação proporcional ao ser aumentada aumentará a velocidade de resposta do sistema de controlo e diminuirá o erro em regime permanente.

Observando a *figura 5* conclui-se que um ganho proporcional muito alto gera um sinal de saída alto, o que pode desestabilizar o sistema, levando a um tempo de estabelecimento maior. Um ganho proporcional muito baixo e o sistema pode falhar em aplicar a ação necessária para corrigir as perturbações.

A ação integral produz um sinal de saída que é proporcional à magnitude e à variação do erro.

$$I_{saida} = K_d \int_0^t e(\tau) d\tau \quad \text{Equação 10}$$

Ação integral corrige o valor da variável manipulada em intervalos regulares, eliminando o erro estacionário e aumentando a resposta do sistema.

Observando a *figura 6* conclui-se que se o ganho integral é baixo o sistema pode levar muito tempo para atingir o valor de referência, contudo, se for elevado o sistema pode se tornar instável.

A ação derivativa produz um sinal de saída que é proporcional à velocidade do erro.

$$D_{saida} = K_d \frac{de^{(t)}}{dt} \qquad \text{Equação 11}$$

Nesta ação, ocorre uma correção antecipada do erro, diminuindo o tempo de resposta e melhorando a estabilidade do sistema.

Observando a *figura 7* conclui-se que a ação derivativa atua em intervalos regulares, havendo o cuidado de não ter muito ruido pois pode tornar o sistema instável e ações de controlo muito lentas pode tornar o sistema instável.

Neste capitulo, explicar-se-á o PID de velocidade e o PID de posição.

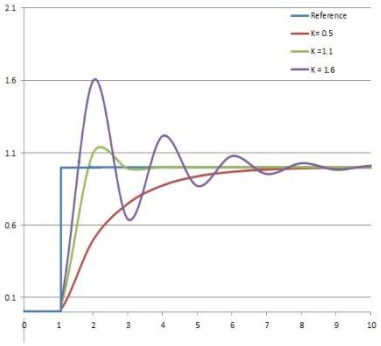


Figura 5 - Ação proporcional

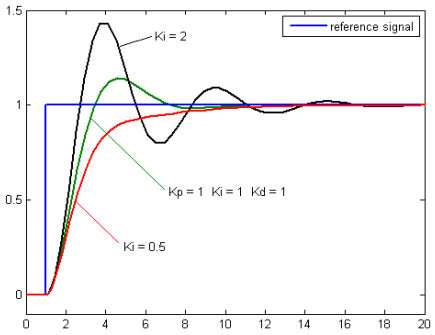


Figura 6 - Ação Integral

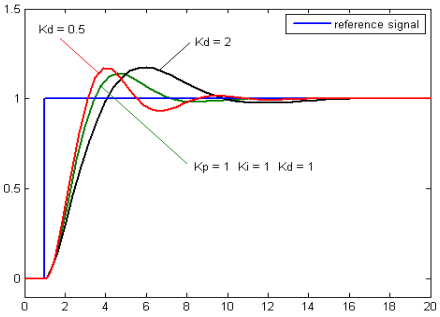


Figura 7 - Ação Derivativa

PID de velocidade

A equação 12 representa o algoritmo de velocidade do PID, calcula-se a variação da saída com o instante anterior.

$$V_n = V_{n-1} + K_p(E_n - E_{n-1}) + K_i E_n + K_d(E_n - 2E_{n-1} + V_{n-2}) \quad \text{Equação 12}$$

Vantagens de PID de Velocidade

- O fenómeno de wind-up (consiste na saturação do controlador através do somatório do erro no tempo integral) não ocorre;
- Não é necessário conhecer o valor da ação de controlo em estado estacionário;

PID de Posição

A equação 13 representa o algoritmo de posição PID, onde cada instante é calculado o valor real do sinal de saída.

$$V_n = K_p \times E_n + K_i \sum_{k=1}^{n} E_k + K_d(E_n - E_{n-1}) \quad \text{Equação 13}$$

No PID de posição ocorre o fenómeno de wind-up, para que o controlador saia da saturação é necessário que o erro troque de sinal durante algum tempo para diminuir o termo integral. Pode-se evitar, se definir-se uma gama de valores aceitável para a variável de saída.

Variáveis do Sistema:

V- Variável de saída;
E-Erro;
K_p – Ganho proporcional;
K_i- Ganho integral;
K_d- Ganho derivativo;

Tabela 2 - Influência dos ganhos no PID

	Rise Time	***Overshoot***	***Setting Time***	e_{rp}
K_p	Diminui	Aumenta	Sem efeito	Diminui
K_i	Diminui	Aumenta	Aumenta	Elimina
K_d	Sem efeito	Diminui	Diminui	Sem efeito

Trabalho Realizado

Neste projeto desenvolveu-se a construção de um robô equilibrista de raiz, desde a sua construção, escolha de um controlo e uma aplicação android.

Estrutura física

O robô construiu-se em madeira, pois sendo um material mais barato e de fácil utilização permitiu que os custos não fossem elevados. Para tal, foi necessária uma pesquisa sobre o intuito de ter o melhor design possível.

Figura 8 - Estrutura final

Figura 9 - Estrutura final com componentes

Parâmetros do robô

Na tabela 3, são apresentados os parâmetros e as medidas do robô.

Tabela 3 - Parâmetros do robô

Parâmetro	**Unidade**	**Valor**
m	[kg]	0.01692
R	[m]	0.0335
M	[kg]	2.96
W	[m]	0.235

D	[m]	0.135
H	[m]	0.31

Descrição do Hardware Implementado

Vários componentes foram utilizados no protótipo, desde dois motores motor, encoder, ponte H, sensor de distancia, um acelerómetro/giroscópio, duas baterias externas e um microcontrolador.

Motor

Para movimentar-se o robô, foram necessários dois motores fornecidos em projeto I, *figura 10*. Esses motores são iguais sendo que foram colocados no robô de maneira a que o mesmo se equilibrasse e movimenta-se.

Figura 10 - SD3757

Encoder

Para obter a posição angular do pêndulo foi usado o encoder fornecido em projeto I, que estava junto ao motor como se pode ver na figura 10. É um componente importante pois só com os valores obtidos por ele é que será possível utilizar o controlo PID de velocidade.

Sensor de distância

Inicialmente pensou-se em utilizar um sensor de ultrassom mas devido ao delay que seria necessário para o poder utilizar, iria perder-se tempo nessa função o que provocaria destabilização do robô.

Figura 11 - Ultrassom

Para prevenir que o robô não fosse contra nenhum obstáculo foi utilizado o sensor de infravermelhos*, figura 11*. Assim, quando detetado algum obstáculo o robô anda para trás para evitar a colisão.

Na tabela 4 encontram-se as especificações e características do sensor de infravermelhos.

Tabela 4 - Especificações e características do sensor de distância

Tensão de Alimentação (V)	5
Corrente de Funcionamento (mA)	30
Alcance Máximo (cm)	80
Alcance Mínimo (cm)	10

Ponte H

A ponte H é um circuito que permite alterar a direção para a qual os motores rodam e ainda aplicar a PWM desejada a cada um

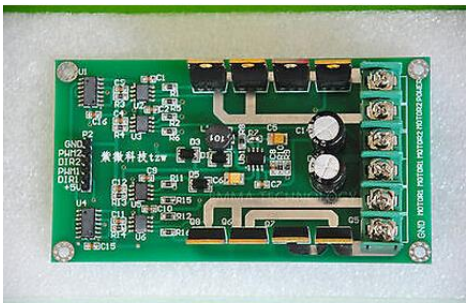


Figura 12 - Ponte H

Microcontrolador

Utilizou-se o Arduino Mega 2560, *figura 13*, para processar a informação proveniente dos sensores e aplicação. Assim, este controla toda a informação proveniente do controlo utilizado e aplica nos devidos componentes.

Figura 13 - Arduino Mega 2560

Bluetooth

Utilizou-se o Bluetooth HC-06*, figura 14*, para comunicar da aplicação com o arduino, utilizando os respetivos pinos RX e TX.

Na tabela 5 encontram-se as especificações e características do bluetooth.

Figura 14 - HC-06

Tabela 5- Especificações e características do Bluetooth

Tensão de Alimentação (V)	3.6
Pinos RX e TX	Comunicação serie

Acelerómetro/giroscópio

Utilizou-se o giroscópio MPU650, *figura 15*, para a leitura da inclinação do robô. Este sensor possui um acelerómetro e um giroscópio cada um com três eixos.

Figura 15 - *MPU650*

Baterias

Utilizou-se duas baterias, *figura 16*, uma para alimentar os motores e outra para alimentar o Arduíno e os restantes componentes.

Figura 16 - Baterias

Aplicação android

Pesquisou-se formas de construir uma aplicação simples e de fácil utilização, para isso, utilizou-se o MIT creator. O layout encontra-se na *figura 17*.

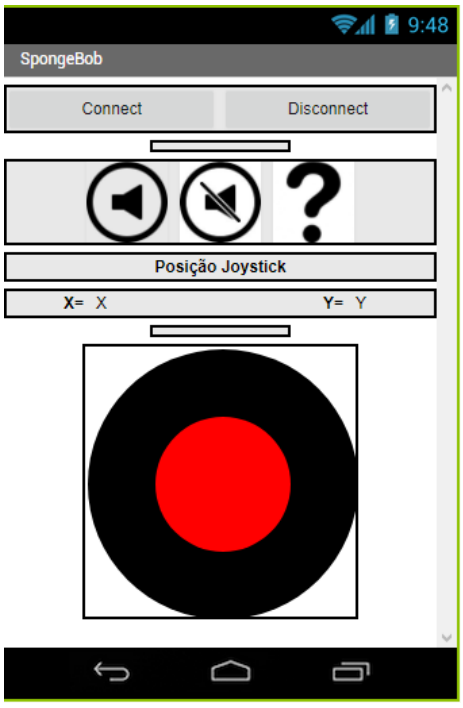


Figura 17 - Layout aplicação

Esta aplicação conecta-se com o Bluetooth, posteriormente retorna o valor do Joystick. O arduino recebe esse valor para poder movimentar o robô para qualquer direção. Para uma análise mais detalhada do código, encontra-se em anexo A.2 o respetivo código de blocos.

Código C Desenvolvido

O código de controlo do sistema foi desenvolvido em linguagem C, que pode ser consultado nos anexos do relatório. O arduino foi a ferramenta que se utilizou no desenvolvimento do código e para o processamento do código o Arduino Mega 2560.

Nesta etapa serão explicadas partes do código fundamentais para o controlo do nosso robô.

PID de Posição

Inicialmente utilizou-se o algoritmo PID de posição, explicado anteriormente. Realizou-se vários testes com este algoritmo e sendo o nosso sistema naturalmente instável verificou-se que provocaria grande esforço ao sistema. Observou-se que para ganhos baixos o robô não se equilibrava, para ganhos altos exigia grande esforço ao sistema e não teria um movimento suave. Assim, conclui-se que a melhor opção seria utilizar o algoritmo de velocidade.

PID de Velocidade

Inicialmente, configurou-se os ganhos do PID de velocidade como poderemos ver na *figura 18.*

```
kp = 7;
ki = 1.20;//valores bons para equilibrar
kd = 0.0630;
```

Figura 18 - Configuração dos ganhos PID velocidade

Assim, em cada iteração retirou-se o erro.

Obteve-se esse erro através da subtração do ângulo de referência pelo ângulo lido do nosso acelerómetro/giroscópio. Posteriormente, tendo em base o erro anterior, calcula-se a nossa força de atuação, descrito na *figura 19*.

```
//*****************************************PID VELOCIDADE para calcular a força para o angulo de inclinacao *****************
   erro = reference_angle - angle_pitch_output; // erro = reference- sensor_value
   Cn = Cn_1+(kp * (erro - erro_anterior_1)) + (ki * erro) + (kd * (erro - (2 * erro_anterior_1) + erro_anterior_2));//+Cn_enc
   Cn2 = Cn_12+(kp * (erro2 - erro_anterior_12)) + (ki * erro2) + (kd * (erro2 - (2 * erro_anterior_12) + erro_anterior_22));/
//*************************************************************************************************************+*****************
```

Figura 19 - Código do PID Velocidade

Por fim, atribuímos ao PWM dos motores essa força de atuação.

Rotinas de interrupção do Encoder

O encoder fornecido, possui dois canais de saída (Fase A e Fase B). Cada fase tem uma interrupção que é ativada sempre que houver uma transição no pino (deteta a transição de 0→1).

O encoder tem uma resolução de 406 pulsos/volta. A *figura 20*, apresenta os sinais de saída das fases A e B em ambos os sentidos (horário e anti-horário).

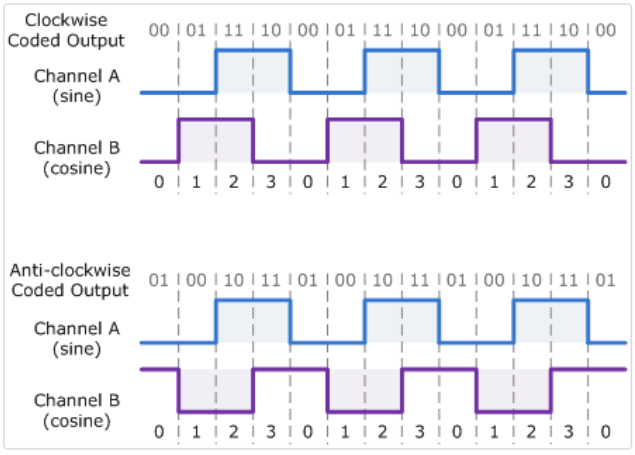


Figura 20 - Sinais de saída do encoder

As rotinas de interrupção relativas as fases A e B encontram-se na *figura 21*. Observando os impulsos que são recebidos, determinou-se a direção do robô, incrementando-se uma variável que declara a distância que o robô se deslocou.

```
void ai0() {
  if (digitalRead(3) == LOW) {
    counter_encoder++;
  } else {
    counter_encoder--;
  }
}

void ai1() {
  if (digitalRead(2) == LOW) {
    counter_encoder--;
  } else {
    counter_encoder++;
  }
}
```

Figura 21 - Algoritmo de interrupção

Rotinas do sensor de distancia

Para calcular a presença de algum obstáculo em frente ao robô é executado código presente na figura 22. Foi necessário aplicar um filtro passa baixo para atenuar e acertar os valores lidos pelo sensor de infravermelhos.

```
// ******Filtro passa baixo para atenuar e acertar os valores lidos do sensor IR**************
  sensor_value = analogRead(sensor);
  sensor_filtrado = sensor_value * 0.1 + sensor_value_ant * 0.9;
  sensor_value_ant = sensor_value;
// ********************************************************************************************************************
```

Figura 22 – Código de leitura dos valores lidos pelo sensor de infravermelho

Assim, após detetar um obstáculo, caso este se encontre a menos de 10 cm o robô irá receber uma força de repulsão aumentando a sua velocidade para o lado oposto do obstáculo.

```
//****************************** Caso encontre obstaculo recua o robo **********************
   erro = (reference_angle) - angle_pitch_output;
 if (sensor_value >= 300)
  {
    erro = (reference_angle-0.332) - angle_pitch_output;
    forca_repulsao = -10;//aumenta velocidade para tras
  }
    else{
          forca_repulsao=0; // se nao encontrar obstaculo nao temos forca repulsao
        if(Cn_enc<0)
            {
               erro = (reference_angle+0.332) - angle_pitch_output; // erro = reference- sensor_value
            }
          else
            {
               erro = (reference_angle-0.121) - angle_pitch_output; // erro = reference- sensor_value
            }
          }
  erro2 = erro;//erro do motor e igual ao 1, pois sao para controlar os 2 ao mesmo tempo
```

Figura 23 - Código de detenção de obstáculos

Resultados

Com o PID de velocidade conseguimos um equilíbrio aceitável, contudo era possível melhorar o equilíbrio se mudássemos os ganhos utilizados. Assim, foram realizadas varias experiencias e testes, sendo que foram selecionados os que tiveram melhores resultados.

Como podemos ver neste primeiro vídeo, anexo A.3 vídeo um, o robô equilibra-se perfeitamente, sem utilizar os encoders e apenas aplicado o PID de velocidade. Neste segundo vídeo, anexo A.3 vídeo dois, a posição será controlada por um PID que terá um peso na variável de controlo. Assim, usando os encoders saberemos a posição a que o robô se encontra. Como podemos observar neste vídeo, os ganhos deveriam ter melhores ajustes para obter melhores resultados. Também é possível observar que embora o controlo não seja muito suave o robô consegue regressar à sua posição inicial.

No terceiro vídeo, anexo A.3 vídeo três, o robô deteta um objeto e afasta-se para trás ate ficar a uma distância segura. Como o objeto esta sempre perto do infravermelho, o robô afasta-se ate conseguir obter a distância de segurança.

No quarto vídeo, anexo A.4 vídeo quatro, pode-se observar que o robô responde ao que o utilizador indica no joystick. Tal como é visível no vídeo o controlo do robô nesta parte não é tão suave como o desejado, isto deve-se ao facto de ainda não termos aplicado um controlo à velocidade de deslocação do robô, mas só a sua posição.

Embora seja possível melhorar o controlo do nosso robô conseguimos resultados positivos em todos os aspetos que nos propusemos a fazer e, conseguimos conclui-los todos.

Conclusão e Perspetivas Futuras

Este relatório divide-se em 3 fases, sendo elas: fundamentos teóricos, trabalho realizado e resultados.

A primeira fase, tem por objetivo fazer um estudo ao sistema proposto. Fazendo a análise matemática do sistema foi possível perceber e escolher o melhor controlo para este caso. Nesta fase ainda foi pesquisada a melhor forma de construir o robô, assim como material a utilizar e quais os componentes necessários para a realização do projeto.

A segunda fase, tem como objetivo principal desenvolver o controlo escolhido anteriormente. Para isso, é necessário construir o robô. Após a conclusão do mesmo, foi necessário o teste de todos os componentes que o robô iria utilizar (como motores etc).

Para se poder controlar a direção do robô, foi criada uma aplicação android. Assim torna-se acessível e fácil mandar o robô para qualquer local. Por fim, para aplicar a navegação autónoma, testou-se os encoders, verificando que um deles estava estragado o que levou ao atraso do projeto.

Depois de vários testes com diferentes tipos de controlo conclui-se que seria necessário utilizar o PID de velocidade pois era o que tinha um controlo mais suave. E mais robusto perante perturbações. Após a etapa anterior, a etapa principal, implementamos os encoders. Nesta fase já era possível o robô retornar à sua posição inicial. Embora esta fase tenha sido concluída o sistema não apresenta um controlo tão suave como o desejado. Na terceiro fase, teríamos de observar a existência de

obstáculos à frente do robô e mais uma vez concluímos esta fase com sucesso. O robô afasta-se dos obstáculos mantendo uma distância de segurança. Por fim, depois de estabelecida a comunicação Bluetooth entre o arduino e a app do telemóvel, o robô seguia as ordens dadas pelo utilizador à medida que este movia o joystick. Embora esta etapa tenha sido alcançada o controlo não era tão suave como desejaríamos e os ganhos precisam de alguns ajustes.

Em suma todas as etapas pretendidas foram alcançadas mas existem bastantes melhorias que podem ser feito para melhorar o nosso trabalho. Uma das melhorias a realizar, seria dimensionar melhor os ganhos para todas as etapas e com isto, obter um controlo mais suave. Embora não fosse objetivo inicialmente, a implementação tardia dos encoders, limitou as nossa opções em termos de controlo. Se tivéssemos mais tempo para concluir o projeto, poderíamos experimentar outros tipos de controlo como colocação de polos, e poderiam ter sido obtidos resultados mais positivos. Na nossa variável de controlo poderíamos ter uma componente para a velocidade e outra para a posição o que tornaria o deslocamento e equilíbrio do robô mais fácil e mais suave. Aumentar a quantidade de sensores infravermelhos para o robô ter uma "visão" de 360º.

Na realidade, existindo imenso espaço para melhoria no nosso robô, estamos satisfeitos por realizar todas as etapas propostas e ter acrescentado algumas ao que tínhamos proposto inicialmente. Agradecemos ao corpo docente por todo o auxilio prestado.

Referências Bibliográficas

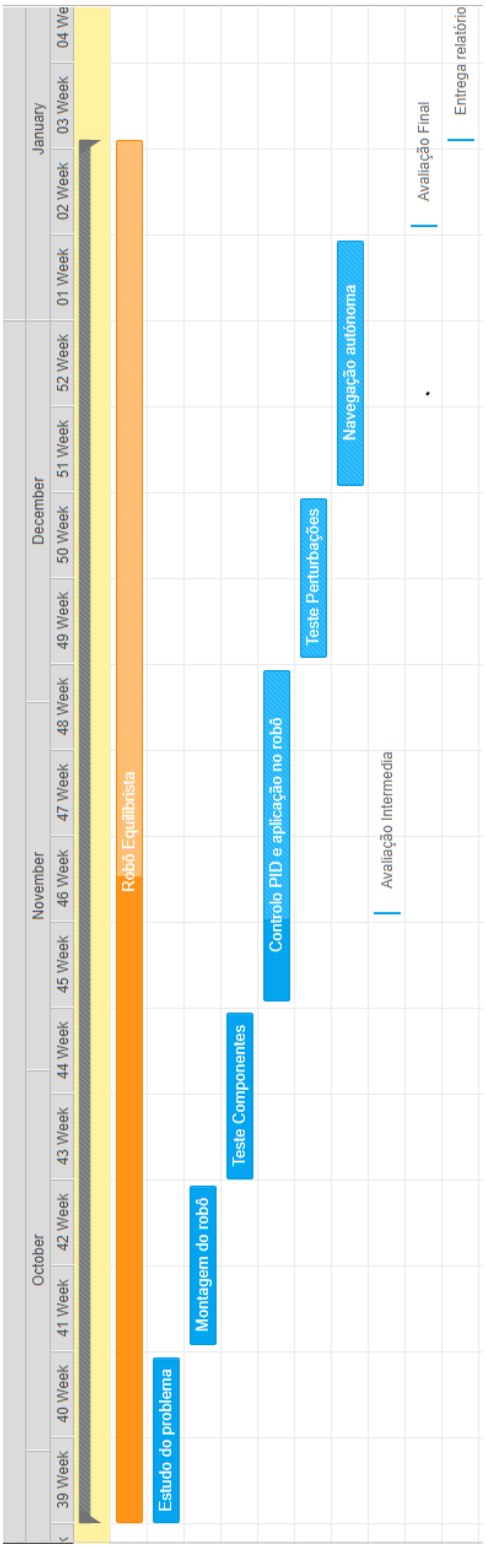
- Wikipédia, “Segway,” [Online]. Available: https://pt.wikipedia.org/wiki/Segway. [Acedido em 20 01 2018].
- “Explicando a Teoria PID,” [Online]. Available: http://www.ni.com/white-paper/3782/pt/#toc1. [Acedido em 20 01 2018].

- Wikipédia, “Controlador proporcional integral derivativo,”[Online]. Available: https://pt.wikipedia.org/wiki/Controlador_proporcional_integral_derivativo [Acedido em 20 01 2018].
- “wowwee,”[Online].Available: https://wowwee.com/mip/ [Acedido em 9 11 2017].
- [Online]. Available: https://elearning.uminho.pt/bbcswebdav/pid-731289-dt-content-rid-1550197_1/courses/1718.9307N7_1/1718.9307N7_1_ImportedContent_20170911115027/Motors_Spur_Gear_DC_Catalogue_1W-40W.pdf [Acedido em 12 11 2017].

Anexos

A.0

De seguida encontra-se a calendarização que foi prevista para todas as tarefas associadas ao desenvolvimento do robô.



A.1

3.1 Two-Wheeled Inverted Pendulum Model

NXTway-GS can be considered as a two wheeled inverted pendulum model shown in Figure 3-1.

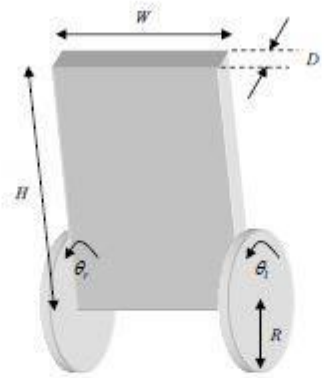
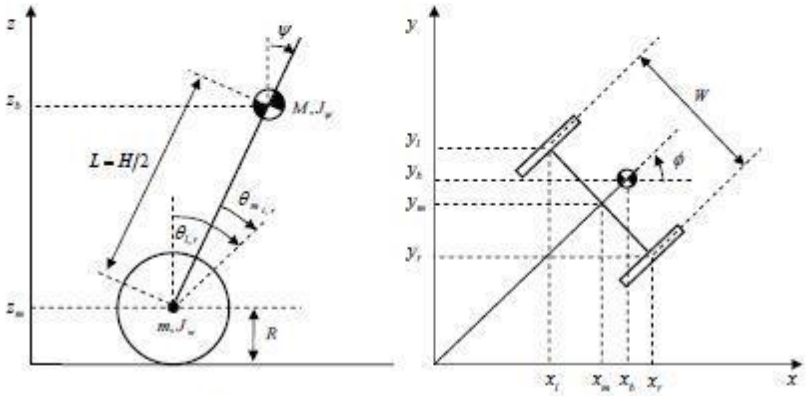


Figure 3-1 Two-wheeled inverted pendulum

Figure 3-2 shows side view and plane view of the two wheeled inverted pendulum. The coordinate system used in 3.2 Motion Equations of Two-Wheeled Inverted Pendulum is described in Figure 3-2.



ψ : body pitch angle $\theta_{l,r}$: wheel angle (l, r indicates left and right) $\theta_{m_{l,r}}$: DC motor angle

Figure 3-2 Side view and plane view of two-wheeled inverted pendulum

Physical parameters of NXTway-GS are the following.

$g = 9.81$	$[m/\sec^2]$	:	Gravity acceleration
$m = 0.03$	$[kg]$	:	Wheel weight
$R = 0.04$	$[m]$	:	Wheel radius
$J_w = mR^2/2$	$[kgm^2]$	:	Wheel inertia moment
$M = 0.6$	$[kg]$	:	Body weight
$W = 0.14$	$[m]$	:	Body width
$D = 0.04$	$[m]$	:	Body depth
$H = 0.144$	$[m]$	:	Body height
$L = H/2$	$[m]$	:	Distance of the center of mass from the wheel axle
$J_p = ML^2/3$	$[kgm^2]$	:	Body pitch inertia moment
$J_\psi = M(W^2 + D^2)/12$	$[kgm^2]$	:	Body yaw inertia moment
$J_m = 1\times10^{-5}$	$[kgm^2]$	:	DC motor inertia moment
$R_m = 6.69$	$[\Omega]$	:	DC motor resistance
$K_b = 0.468$	$[V\sec/rad]$	:	DC motor back EMF constant
$K_t = 0.317$	$[Nm/A]$	:	DC motor torque constant
$n = 1$		:	Gear ratio
$f_m = 0.0022$		:	Friction coefficient between body and DC motor
$f_w = 0$		:	Friction coefficient between wheel and floor.

- We use the values described in reference [2] for R_m, K_b, K_t.
- We use the values that seems to be appropriate for J_m, n, f_m, f_w, because it is difficult to measure.

3.2 Motion Equations of Two-Wheeled Inverted Pendulum

We can derive motion equations of two-wheeled inverted pendulum by the Lagrangian method based on the coordinate system in Figure 3-2. If the direction of two-wheeled inverted pendulum is x-axis positive direction at $t = 0$, each coordinates are given as the following.

$$(\theta,\ \phi) = \left(\frac{1}{2}(\theta_r + \theta_l)\ \ \frac{R}{W}(\theta_r - \theta_l)\right) \tag{3.1}$$

$$(x_m,\ y_m,\ z_m) = \left(\int \dot{x}_m dt,\ \int \dot{y}_m dt,\ R\right),\ (\dot{x}_m,\ \dot{y}_m) = \left(R\dot{\theta}\cos\phi\ \ R\dot{\theta}\sin\phi\right) \tag{3.2}$$

$$(x_l,\ y_l,\ z_l) = \left(x_m - \frac{W}{2}\sin\phi,\ y_m + \frac{W}{2}\cos\phi,\ z_m\right) \tag{3.3}$$

$$(x_r,\ y_r,\ z_r) = \left(x_m + \frac{W}{2}\sin\phi,\ y_m - \frac{W}{2}\cos\phi,\ z_m\right) \tag{3.4}$$

$$(x_b,\ y_b,\ z_b) = (x_m + L\sin\psi\cos\phi,\ y_m + L\sin\psi\sin\phi,\ z_m + L\cos\psi) \tag{3.5}$$

The translational kinetic energy T_1, the rotational kinetic energy T_2, the potential energy U are

$$T_1 = \frac{1}{2}m\left(\dot{x}_l^2 + \dot{y}_l^2 + \dot{z}_l^2\right) + \frac{1}{2}m\left(\dot{x}_r^2 + \dot{y}_r^2 + \dot{z}_r^2\right) + \frac{1}{2}M\left(\dot{x}_b^2 + \dot{y}_b^2 + \dot{z}_b^2\right) \tag{3.6}$$

$$T_2 = \frac{1}{2}J_w\dot{\theta}_l^2 + \frac{1}{2}J_w\dot{\theta}_r^2 + \frac{1}{2}J_\psi\dot{\psi}^2 + \frac{1}{2}J_\phi\dot{\phi}^2 + \frac{1}{2}n^2J_m(\dot{\theta}_l - \dot{\psi})^2 + \frac{1}{2}n^2J_m(\dot{\theta}_r - \dot{\psi})^2 \tag{3.7}$$

$$U = mgz_l + mgz_r + Mgz_b \tag{3.8}$$

The fifth and sixth term in T_2 are rotation kinetic energy of an armature in left and right DC motor. The Lagrangian L has the following expression.

$$L = T_1 + T_2 - U \tag{3.9}$$

We use the following variables as the generalized coordinates.

θ : Average angle of left and right wheel
ψ : Body pitch angle
ϕ : Body yaw angle

Lagrange equations are the following

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) - \frac{\partial L}{\partial \theta} = F_\theta \tag{3.10}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\psi}}\right) - \frac{\partial L}{\partial \psi} = F_\psi \tag{3.11}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\phi}}\right) - \frac{\partial L}{\partial \phi} = F_\phi \tag{3.12}$$

We derive the following equations by evaluating Eqs. (3.10) - (3.12).

$$\left\{(2m+M)R^2 + 2J_w + 2n^2J_m\right\}\ddot{\theta} + \left(MLR\cos\psi - 2n^2J_m\right)\ddot{\psi} - MLR\dot{\psi}^2\sin\psi = F_\theta \tag{3.13}$$

$$\left(MLR\cos\psi - 2n^2J_m\right)\ddot{\theta} + \left(ML^2 + J_\psi + 2n^2J_m\right)\ddot{\psi} - MgL\sin\psi - ML^2\dot{\phi}^2\sin\psi\cos\psi = F_\psi \tag{3.14}$$

$$\left[\frac{1}{2}mW^2 + J_\phi + \frac{W^2}{2R^2}\left(J_w + n^2J_m\right) + ML^2\sin^2\psi\right]\ddot{\phi} + 2ML^2\dot{\psi}\dot{\phi}\sin\psi\cos\psi = F_\phi \tag{3.15}$$

In consideration of DC motor torque and viscous friction, the generalized forces are given as the following

$$(F_\theta,\ F_\psi,\ F_\phi) = \left(F_l + F_r,\ F_\psi,\ \frac{W}{2R}(F_r - F_l)\right) \tag{3.16}$$

$$F_l = nK_t i_l + f_m(\dot{\psi} - \dot{\theta}_l) - f_w\dot{\theta}_l \tag{3.17}$$

$$F_r = nK_t i_r + f_m(\dot{\psi} - \dot{\theta}_r) - f_w\dot{\theta}_r \tag{3.18}$$

$$F_\psi = -nK_t i_l - nK_t i_r - f_m(\psi - \theta_l) - f_m(\psi - \theta_r) \tag{3.19}$$

where $i_{l,r}$ is the DC motor current.

We cannot use the DC motor current directly in order to control it because it is based on PWM (voltage) control. Therefore, we evaluate the relation between current $i_{l,r}$ and voltage $v_{l,r}$ using DC motor equation. If the friction inside the motor is negligible, the DC motor equation is generally as follows

$$L_m\dot{i}_{l,r} = v_{l,r} + K_b(\dot{\psi} - \dot{\theta}_{l,r}) - R_m i_{l,r} \tag{3.20}$$

Here we consider that the motor inductance is negligible and is approximated as zero. Therefore the current is

$$i_{l,r} = \frac{v_{l,r} + K_b(\dot{\psi} - \dot{\theta}_{l,r})}{R_m} \tag{3.21}$$

From Eq.(3.21), the generalized force can be expressed using the motor voltage.

$$F_\theta = \alpha(v_l + v_r) - 2(\beta + f_w)\theta + 2\beta\psi \tag{3.22}$$

$$F_\psi = -\alpha(v_l + v_r) + 2\beta\theta - 2\beta\psi \tag{3.23}$$

$$F_\phi = \frac{W}{2R}\alpha(v_r - v_l) - \frac{W^2}{2R^2}(\beta + f_w)\dot{\phi} \tag{3.24}$$

$$\alpha = \frac{nK_t}{R_m},\quad \beta = \frac{nK_tK_b}{R_m} + f_m \tag{3.25}$$

3.3 State Equations of Two-Wheeled Inverted Pendulum

We can derive state equations based on modern control theory by linearizing motion equations at a balance point of NXTway-GS. It means that we consider the limit $\psi \to 0$ ($\sin\psi \to \psi$, $\cos\psi \to 1$) and neglect the second order term like $\dot{\psi}^2$. The motion equations (3.13) – (3.15) are approximated as the following

$$\left[(2m+M)R^2 + 2J_w + 2n^2J_m\right]\ddot{\theta} + \left(MLR - 2n^2J_m\right)\ddot{\psi} = F_\theta \tag{3.26}$$

$$\left(MLR - 2n^2J_m\right)\ddot{\theta} + \left(ML^2 + J_\psi + 2n^2J_m\right)\ddot{\psi} - MgL\psi = F_\psi \tag{3.27}$$

$$\left[\frac{1}{2}mW^2 + J_\phi + \frac{W^2}{2R^2}\left(J_w + n^2J_m\right)\right]\ddot{\phi} = F_\phi \tag{3.28}$$

Eq. (3.26) and Eq. (3.27) has θ and ψ, Eq. (3.28) has ϕ only. These equations can be expressed in the form

$$E\begin{bmatrix}\ddot{\theta}\\ \ddot{\psi}\end{bmatrix} + F\begin{bmatrix}\dot{\theta}\\ \dot{\psi}\end{bmatrix} + G\begin{bmatrix}\theta\\ \psi\end{bmatrix} = H\begin{bmatrix}v_l\\ v_r\end{bmatrix} \tag{3.29}$$

$$E = \begin{bmatrix}(2m+M)R^2 + 2J_w + 2n^2J_m & MLR - 2n^2J_m\\ MLR - 2n^2J_m & ML^2 + J_\psi + 2n^2J_m\end{bmatrix}$$

$$F = 2\begin{bmatrix}\beta + f_w & -\beta\\ -\beta & \beta\end{bmatrix}$$

$$G = \begin{bmatrix}0 & 0\\ 0 & -MgL\end{bmatrix}$$

$$H = \begin{bmatrix}\alpha & \alpha\\ -\alpha & -\alpha\end{bmatrix}$$

$$I\ddot{\phi} + J\dot{\phi} = K(v_r - v_l) \tag{3.30}$$

$$I = \frac{1}{2}mW^2 + J_\phi + \frac{W^2}{2R^2}\left(J_w + n^2J_m\right)$$

$$J = \frac{W^2}{2R^2}(\beta + f_w)$$

$$K = \frac{W}{2R}\alpha$$

Here we consider the following variables $\mathbf{x}_1, \mathbf{x}_2$ as state, and $\mathbf{u}$ as input. $\mathbf{x}^T$ indicates transpose of $\mathbf{x}$.

$$\mathbf{x}_1 = [\theta, \quad \psi, \quad \dot{\theta}, \quad \dot{\psi}]^T, \quad \mathbf{x}_2 = [\phi, \quad \dot{\phi}]^T, \quad \mathbf{u} = [v_l, \quad v_r]^T \tag{3.31}$$

Consequently, we can derive state equations of two-wheeled inverted pendulum from Eq. (3.29) and Eq. (3.30).

$$\dot{\mathbf{x}}_1 = A_1\mathbf{x}_1 + B_1\mathbf{u} \tag{3.32}$$

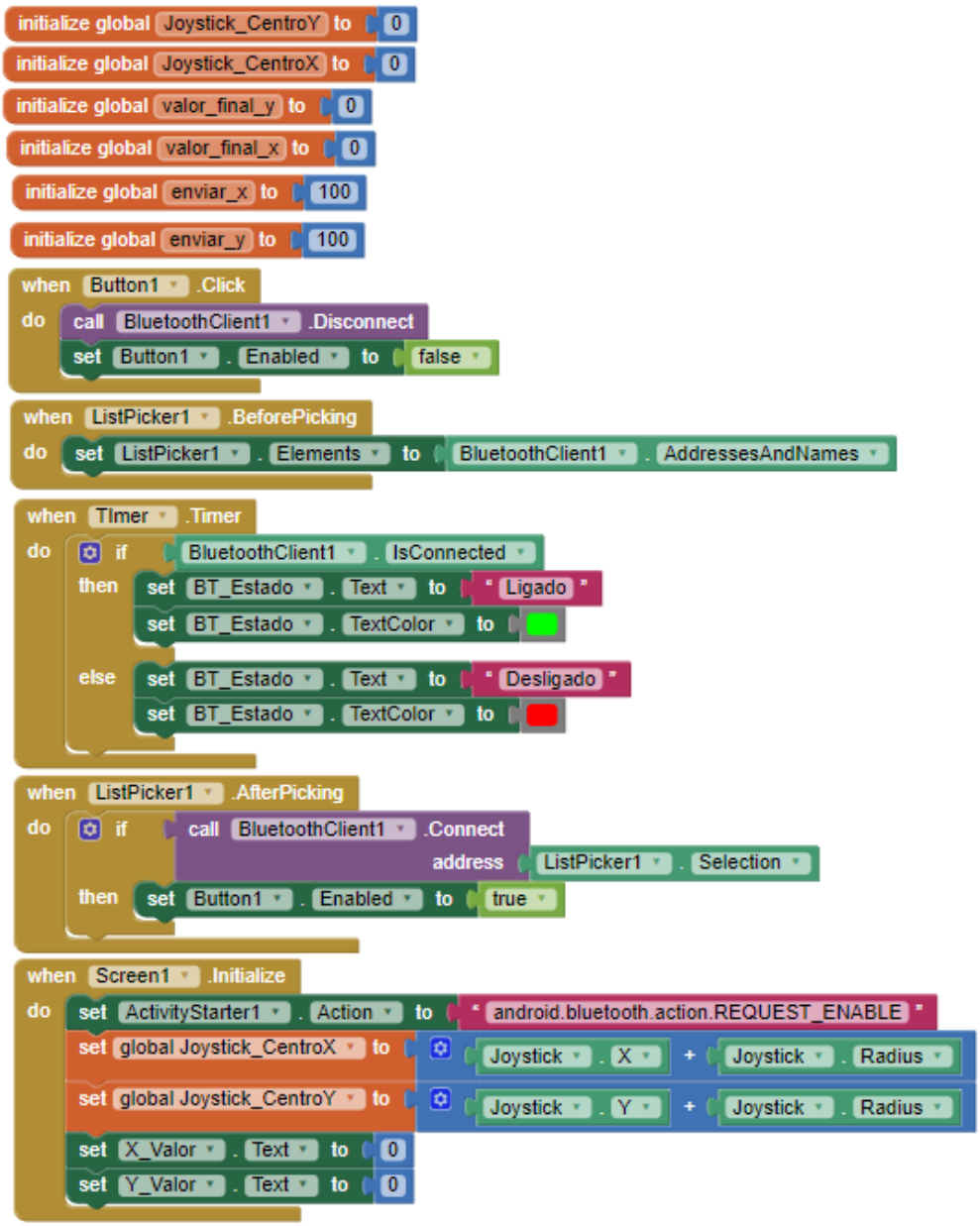
$$\dot{\mathbf{x}}_2 = A_2\mathbf{x}_2 + B_2\mathbf{u} \tag{3.33}$$

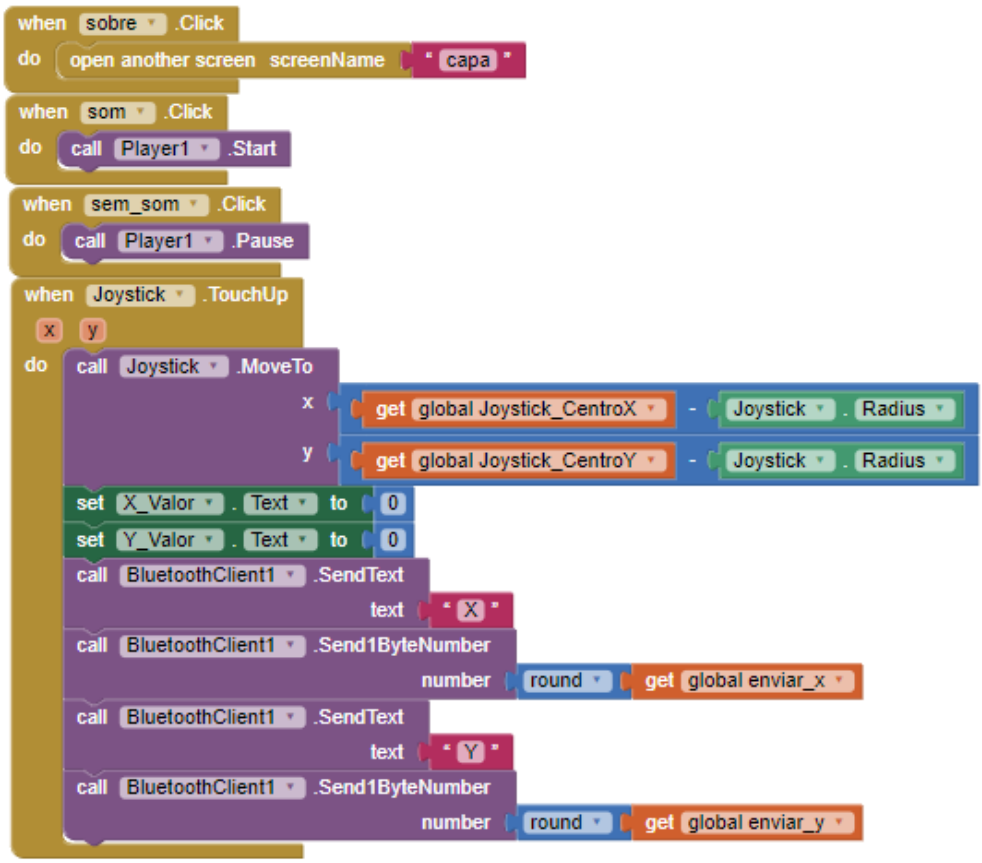
$$A_1 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & A_1(3,2) & A_1(3,3) & A_1(3,4) \\ 0 & A_1(4,2) & A_1(4,3) & A_1(4,4) \end{bmatrix}, \quad B_1 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ B_1(3) & B_1(3) \\ B_1(4) & B_1(4) \end{bmatrix} \tag{3.34}$$

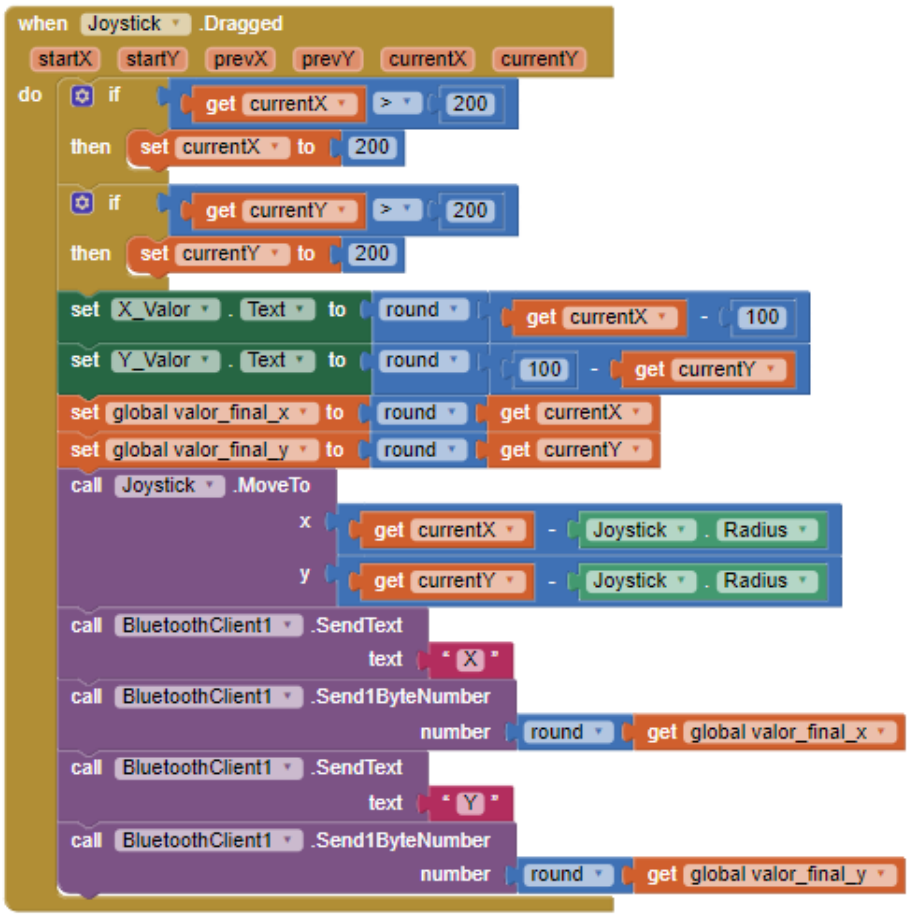
$$A_2 = \begin{bmatrix} 0 & 1 \\ 0 & -J/I \end{bmatrix}, \quad B_2 = \begin{bmatrix} 0 & 0 \\ -K/I & K/I \end{bmatrix} \tag{3.35}$$

$$A_1(3,2) = -gMLE(1,2)/\det(E)$$
$$A_1(4,2) = gMLE(1,1)/\det(E)$$
$$A_1(3,3) = -2[(\beta + f_w)E(2,2) + \beta E(1,2)]/\det(E)$$
$$A_1(4,3) = 2[(\beta + f_w)E(1,2) + \beta E(1,1)]/\det(E)$$
$$A_1(3,4) = 2\beta[E(2,2) + E(1,2)]/\det(E)$$
$$A_1(4,4) = -2\beta[E(1,1) + E(1,2)]/\det(E)$$
$$B_1(3) = \alpha[E(2,2) + E(1,2)]/\det(E)$$
$$B_1(4) = -\alpha[E(1,1) + E(1,2)]/\det(E)$$
$$\det(E) = E(1,1)E(2,2) - E(1,2)^2$$

A.2


initialize global Joystick_CentroY to 0
initialize global Joystick_CentroX to 0
initialize global valor_final_y to 0
initialize global valor_final_x to 0
initialize global enviar_x to 100
initialize global enviar_y to 100
when Button1 .Click
do call BluetoothClient1 .Disconnect
set Button1 . Enabled to false
when ListPicker1 .BeforePicking
do set ListPicker1 . Elements to BluetoothClient1 . AddressesAndNames
when TImer .Timer
do if BluetoothClient1 . IsConnected
then set BT_Estado . Text to " Ligado "
set BT_Estado . TextColor to
else set BT_Estado . Text to " Desligado "
set BT_Estado . TextColor to
when ListPicker1 .AfterPicking
do if call BluetoothClient1 .Connect
address ListPicker1 . Selection
then set Button1 . Enabled to true
when Screen1 .Initialize
do set ActivityStarter1 . Action to " android.bluetooth.action.REQUEST_ENABLE "
set global Joystick_CentroX to Joystick . X + Joystick . Radius
set global Joystick_CentroY to Joystick . Y + Joystick . Radius
set X_Valor . Text to 0
set Y_Valor . Text to 0


when sobre .Click
do open another screen screenName " capa "
when som .Click
do call Player1 .Start
when sem_som .Click
do call Player1 .Pause
when Joystick .TouchUp
x y
do call Joystick .MoveTo
x get global Joystick_CentroX - Joystick . Radius
y get global Joystick_CentroY - Joystick . Radius
set X_Valor . Text to 0
set Y_Valor . Text to 0
call BluetoothClient1 .SendText
text " X "
call BluetoothClient1 .Send1ByteNumber
number round get global enviar_x
call BluetoothClient1 .SendText
text " Y "
call BluetoothClient1 .Send1ByteNumber
number round get global enviar_y


when Joystick .Dragged
startX startY prevX prevY currentX currentY
do if get currentX > 200
then set currentX to 200
if get currentY > 200
then set currentY to 200
set X_Valor . Text to round get currentX - 100
set Y_Valor . Text to round 100 - get currentY
set global valor_final_x to round get currentX
set global valor_final_y to round get currentY
call Joystick .MoveTo
x get currentX - Joystick . Radius
y get currentY - Joystick . Radius
call BluetoothClient1 .SendText
text " X "
call BluetoothClient1 .Send1ByteNumber
number round get global valor_final_x
call BluetoothClient1 .SendText
text " Y "
call BluetoothClient1 .Send1ByteNumber
number round get global valor_final_y

A.3

1. Vídeo um: https://youtu.be/wI5memg-8OI
2. Vídeo dois: https://www.youtube.com/watch?v=ugGCVEc5Y2U&feature=youtu.be
3. Vídeo três:https://youtu.be/2Vcas5Psg8U;
4. Vídeo quatro: https://youtu.be/tXBSjIbV0jg

A.4

O Código utilizado no rôbo:

```
#include "TimerOne.h"
#include <Wire.h>

#define PWM1 7  //PWM motor 1
#define DIR1 6 // Direction of Motor 1
#define PWM2 8  // PWM motor 2
#define DIR2 9 // Direction of Motor 2
#define PIN_TRIG 36
#define PIN_ECHO 34
#define MAX_ADC 1023 //max 10 bit value of the ADC
#define sensor A0 // Sharp IR GP2Y0A41SK0F (4-30cm, analog)

int counter_encoder=0;//variavel do encoder
unsigned char byte_anterior;//variaveis para o bluetooth
unsigned char byte_atual; //variaveis para o bluetooth
unsigned char valor_x;//variaveis para o bluetooth
unsigned char valor_y;//variaveis para o bluetooth
int gyro_x, gyro_y, gyro_z;
long acc_x, acc_y, acc_z, acc_total_vector;
int temperature;
long loop_timer;
//ganles, and accelarations
float angle_pitch, angle_roll;
boolean set_gyro_angles;
float angle_roll_acc, angle_pitch_acc;
float angle_pitch_output, angle_roll_output;
// variaveis para o controlo
int reference_zero = 30; // value in degres from when the robot is leveled
float reference_angle; //valor correspondente ao valor que queremos como referecia
float reference_encoder=0;
volatile float Cn = 0.0; //controlador
volatile float Cn_1 = 0.0;
volatile float Cn2 = 0.0; //controlador
volatile float Cn_12 = 0.0;
volatile float Cn_enc= 0.0;
volatile float Cn_enc_1 = 0.0;
volatile float U1 = 0.0; //variavel de controlo motor 1
volatile float U2 = 0.0; //variavel de controlo motor 2
//ganhso do nosso sistema
float kp; //ganho do controlador proporcional
float ki;//ganho do controloador integrativo
float kd;//ganhodo controlador derivativo
float kp_enc; //ganho do controlador proporcional
float ki_enc;//ganho do controloador integrativo
float kd_enc;//ganhodo controlador derivativo
//variaveis de erro
float erro = 0.0; //erro entre a referencia e valor real
float erro_anterior_1 = 0.0;
float erro_anterior_2 = 0.0;
float erro2 = 0.0; //erro entre a referencia e valor real
float erro_anterior_12 = 0.0;
float erro_anterior_22 = 0.0;
float erro_enc = 0.0; //erro entre a referencia e valor real
float erro_enc_anterior_1 = 0.0;
float erro_enc_anterior_2 = 0.0;
//variaveis para o sensor de distancia
float alpha_repulsor = 0; //funciona mas cai depois
//float alpha_repulsor=0.25;//funciona porreirinho
float distancia_base = 250.0;
float sensor_value_ant = 0;
float sensor_value = 0;

//variaveis para controlar o joystick
float velocidade_x=0;
float velocidade_y=0;
float ganho_velocidade_app_x=0.008;
float ganho_velocidade_app_y=0.18;
volatile char flag_joy=0;
float tolerancia=0.75;
int  erro_ki=erro_ki+erro;
int  erro_kd_anterior=erro;
```

```
//*****************************Defenicoes de variaveis atribuicao de portos e setup dos registos************************
void setup() {
  setup_mpu_6050_registers();                           //Setup the registers of the MPU-6050 (500dfs / +/-8g) and start the gyro
  TCCR4B = TCCR4B & 0b11111000 | 0b1;//muda frequencia de PWM
  pinMode(PWM1, OUTPUT);// define pin as an output
  pinMode(DIR1, OUTPUT);// define port as an output
  pinMode(PWM2, OUTPUT);// define pin as an output
  pinMode(DIR2, OUTPUT);// define port as an output

  Wire.begin();                                         //Start I2C as master
  Serial.begin(115200);                                 //Use only for debugging
  Serial1.begin(9600);

  digitalWrite(2, HIGH);      // turn on pullup resistors
  digitalWrite(3, HIGH);      // turn on pullup resistors
  digitalWrite(DIR1, HIGH); //ESCREVE 1 nos motores por defeito
  digitalWrite(DIR2, HIGH); //pode alterar.se

  Timer1.initialize(4005); // Inicializa o Timer1 e configura para um perido de 0,5 segundos
  Timer1.attachInterrupt(callback); // Configura a funcaoo callback() como a interrupcao do Timer1

//Setting up interrupt
  //A rising pulse from encodenren activated ai0(). AttachInterrupt 0 is DigitalPin nr 2 on moust Arduino.~
   attachInterrupt(0, ai0, RISING);
  //B rising pulse from encodenren activated ai1(). AttachInterrupt 1 is DigitalPin nr 3 on moust Arduino.
  attachInterrupt(1, ai1, RISING);

  kp = 7;
  ki = 1.20;//valores bons para equilibrar
  kd = 0.0630;

  kp_enc = 0.261;
  ki_enc = 0.330;//valores bons para equilibrar
  kd_enc = 0.00560;

  //reference_angle = 4.28; //queremos o angulo 0.0 para equilibrar (quanto maio for este valor, mais inclinado esta para  frente
  reference_angle = 4.29;
}
//**************Interrupcoes para verificar o encoder da roda azul **********************
void ai0() {// Canal A
  // ai0 is activated if DigitalPin nr 2 is going from LOW to HIGH
  // Check pin 3 to determine the direction
  if (digitalRead(3) == LOW) {
    counter_encoder++;
  } else {
    counter_encoder--;
  }
}
void ai1() {//Canal B
  // ai0 is activated if DigitalPin nr 3 is going from LOW to HIGH
  // Check with pin 2 to determine the direction
  if (digitalRead(2) == LOW) {
    counter_encoder--;
  } else {
    counter_encoder++;
  }
}
//**********************Interrupcao timer para ler o giroscopio e accelarometros
//*******************************4 EM 4 ms************************************************
void callback()
{
  //Gyro angle calculations
  //0.0000611 = 1 / (250Hz / 65.5)     // we are working in the 250HZ frequency
  angle_pitch += gyro_x * 0.0000611;                    //Calculate the traveled pitch angle and add this to the angle_pitch variable
  angle_roll += gyro_y * 0.0000611;                     //Calculate the traveled roll angle and add this to the angle_roll variable

  //0.000001066 = 0.0000611 * (3.142(PI) / 180degr) The Arduino sin function is in radians
  angle_pitch += angle_roll * sin(gyro_z * 0.000001066);            //If the IMU has yawed transfer the roll angle to the pitch angel
  angle_roll -= angle_pitch * sin(gyro_z * 0.000001066);            //If the IMU has yawed transfer the pitch angle to the roll angel

  //Accelerometer angle calculations
  acc_total_vector = sqrt((acc_x * acc_x) + (acc_y * acc_y) + (acc_z * acc_z)); //Calculate the total accelerometer vector
  //57.296 = 1 / (3.142 / 180) The Arduino asin function is in radians
  angle_pitch_acc = asin((float)acc_y / acc_total_vector) * 57.296;    //Calculate the pitch angle
  angle_roll_acc = asin((float)acc_x / acc_total_vector) * -57.296;    //Calculate the roll angle

  //Place the MPU-6050 spirit level and note the values in the following two lines for calibration
  angle_pitch_acc -= 0.0;                               //Accelerometer calibration value for pitch
  angle_roll_acc -= 0.0;                                //Accelerometer calibration value for roll

  if (set_gyro_angles) {                                //If the IMU is already started
    angle_pitch = angle_pitch * 0.9900 + angle_pitch_acc * 0.01;     //Correct the drift of the gyro pitch angle with the accelerometer pitch angle
    angle_roll = angle_roll * 0.9900 + angle_roll_acc * 0.01;
  }
  else {                                                //At first start
    angle_pitch = angle_pitch_acc;                      //Set the gyro pitch angle equal to the accelerometer pitch angle
    angle_roll = angle_roll_acc;                        //Set the gyro roll angle equal to the accelerometer roll angle
    set_gyro_angles = true;                             //Set the IMU started flag
  }

  //To dampen the pitch and roll angles a complementary filter is used
  angle_pitch_output = angle_pitch_output * 0 + angle_pitch * 1;   //Take 90% of the output pitch value and add 10% of the raw pitch value
  angle_roll_output = angle_roll_output * 0 + angle_roll * 1;      //Take 90% of the output roll value and add 10% of the raw roll value
}
```

```
//*********************       lOOOP PRICIPAL  // MAIN       ***************************
void loop() {
  static double voltas=0.0;
  float forca_repulsao = 0;
  float forca_repulsao_u1 = 0;
  float forca_repulsao_u2 = 0;
  float sensor_filtrado = 0;
  read_mpu_6050_data();   //Read the raw acc and gyro data from the MPU-6050

  //voltas=counter_encoder/406;valor para 1 volta na roda
  //voltas=counter_encoder/50;//valor para dividir a roda em 8
  voltas=counter_encoder/101.5; //roda dividida em 4
  erro_enc = reference_encoder + voltas; // erro = reference+ sensor_value

// ***********************************************************************************************
// ******Filtro passa baixo para atenuar e acertar os valores lidos do sensor IR**************
  sensor_value = analogRead(sensor);
  sensor_filtrado = sensor_value * 0.01 + sensor_value_ant * 0.99;
  sensor_value_ant = sensor_value;
// ***********************************************************************************************

// ***********************************************************************************************
//****Se o robo na estiver na posicao incial altera os valores nessecarios para se dirigir a origem***************
//********altera angulo de inclinacao para andar para a frente sem tombar*****************
//****************************** Caso encontre obstaculo recua o robo **********************
   erro = (reference_angle) - angle_pitch_output;
 if (sensor_value >= 300)
  {
    erro = (reference_angle-0.332) - angle_pitch_output;
    erro2 = erro;//erro do motor e igual ao 1, pois sao para controlar os 2 ao mesmo tempo
    forca_repulsao = -10;//aumenta velocidade para tras
  }
    else{
    //esta parte a baixo comentada é relativa ao controlo de posicao
      /*   forca_repulsao=0; // se nao encontrar obstaculo nao temos forca repulsao
       if(Cn_enc<0)
          {
            erro = (reference_angle+0.332) - angle_pitch_output; // erro = reference- sensor_value
            erro2 = erro;//erro do motor e igual ao 1, pois sao para controlar os 2 ao mesmo tempo
            forca_repulsao=0;
            forca_repulsao_u1= 0;
            forca_repulsao_u2= 0;
          }
        else
          {
            erro = (reference_angle-0.121) - angle_pitch_output; // erro = reference- sensor_value
            erro2 = erro;//erro do motor e igual ao 1, pois sao para controlar os 2 ao mesmo tempo
            forca_repulsao=0;
            forca_repulsao_u1= 0;
            forca_repulsao_u2= 0;
          }
          */
  //****************************** se ouver comandos para andar da app **********+
if(flag_joy==1) //(erro<tolerancia||erro<-tolerancia)&&
{
  if(velocidade_x==0){
    forca_repulsao_u1= 0;
    forca_repulsao_u2= 0;
  }
  if(velocidade_y==0){
    forca_repulsao_u1= 0;
    forca_repulsao_u2= 0;
  }
   if(velocidade_y>0){
        erro = (reference_angle+0.831) - angle_pitch_output;//inclina robo
        erro2 = erro;//erro do motor e igual ao 1, pois sao para controlar os 2 ao mesmo tempo
        forca_repulsao = -velocidade_y;//aumenta velocidade para frente
        forca_repulsao_u1= 0;
         forca_repulsao_u2= 0;
    }
    if(velocidade_y<0){
       erro = (reference_angle-0.221) - angle_pitch_output;//inclina robo
       erro2 = erro;//erro do motor e igual ao 1, pois sao para controlar os 2 ao mesmo tempo
       forca_repulsao = velocidade_y;//aumenta velocidade para tras
       forca_repulsao_u1= 0;
       forca_repulsao_u2= 0;
    }
     if(velocidade_x>0 && velocidade_y>0) {
          forca_repulsao_u2= -velocidade_x;
          }//aumenta velocidade para tras
     if(velocidade_x<0 && velocidade_y>0){
          forca_repulsao_u1= velocidade_x;
          }
     if(velocidade_x>0 && velocidade_y<0) {~
          forca_repulsao_u2= velocidade_x;
          }//aumenta velocidade para tras
     if(velocidade_x<0 && velocidade_y<0){
          forca_repulsao_u1= -velocidade_x;
          }

}
//  erro2 = erro;//erro do motor e igual ao 1, pois sao para controlar os 2 ao mesmo temp
//***********************************************************************************************
```

```
//*****************************************PID VELOCIDADE para calcular a força para a posicao ******************************
  Cn_enc=Cn_enc_1+(kp_enc * (erro_enc - erro_enc_anterior_1)) + (ki_enc * erro_enc) + (kd_enc * (erro_enc - (2 * erro_enc_anterior_1) + erro_enc_anterior_2)); //formula
PID para controlo de velocidade
//*************************************************************************************************************************************************

//*****************************************PID VELOCIDADE para calcular a força para o angulo de inclinacao ********************
   Cn = Cn_1+(kp * (erro - erro_anterior_1)) + (ki * erro) + (kd * (erro - (2 * erro_anterior_1) + erro_anterior_2));//+forca_repulsao;//formula PID para controlo de velocidade
   Cn2 = Cn_12+(kp * (erro2 - erro_anterior_12)) + (ki * erro2) + (kd * (erro2 - (2 * erro_anterior_12) + erro_anterior_22));//+forca_repulsao;//formula PID para controlo de
velocidade
//*************************************************************************************************************************+*********************************

//***************************Verificacao do maximo permitido nas variaveis*****************
//*************************Barramos os valores*****************************************
  if (Cn >= 254) Cn = 254;
  if (Cn <= -254) Cn = -254;
  if (Cn2 >= 254) Cn2 = 254;
  if (Cn2 <= -254) Cn2 = -254;
  if (Cn_enc > 1) Cn_enc = 1;
  if (Cn_enc <= -1) Cn_enc = -1;
  //*****************************************************************************************
  //*****************************************************************************************
  // ****************************actualizacao das variavveis ******************************
  Cn_1=Cn;
  Cn_12=Cn2;
  erro_anterior_2 = erro_anterior_1; //ACTUALIZA os valores
  erro_anterior_22 = erro_anterior_12; //ACTUALIZA os valores
  erro_anterior_1 = erro; //reload do valor do erro anterior, que agora ÃƒÂ© o atual
  erro_anterior_12 = erro2; //reload do valor do erro anterior, que agora ÃƒÂ© o atual
  Cn_enc_1=Cn_enc;
  erro_enc_anterior_2 = erro_enc_anterior_1;
  erro_enc_anterior_1 = erro_enc;
//*****************************************************************************************

//*******************************
  U1=Cn+forca_repulsao+forca_repulsao_u1;//+Cn_enc;
  U2=Cn2+forca_repulsao+forca_repulsao_u2;//+Cn_enc;

  if (U1 >= 254)  U1 = 254;
  if (U1 <= -254) U1 = -254;
  if (U2 >= 254)  U2 = 254;
  if (U2 <= -254) U2 = -254;

// ***************************************************************************************************
// ***Verifica se os 2 motores recebem o mesmo sinal e trata os dados confome isso**********
// ***************************************************************************************************
//Serial.println("U1");
//Serial.println(U1);
//Serial.println(U2);
  if ( U1 < 0 && U2 < 0)
  {
    digitalWrite(DIR1, HIGH); //ESCREVE 0 na ponte H para os motores andarem num sentido
    digitalWrite(DIR2, HIGH);// depois de verificar a rotacao dos motores podemos alterara esta variavel
    U1 = U1 * -1;
    U2 = U2 * -1;
  }
  else if(U1>=0 && U2>=0 ){
    digitalWrite(DIR1, LOW); //Se erro por positivo os motores andam para o outro lado
    digitalWrite(DIR2, LOW);
  }
  else if ( U1 < 0 && U2 > 0)
  {
    digitalWrite(DIR1, HIGH); //ESCREVE 0 na ponte H para os motores andarem num sentido
    digitalWrite(DIR2, LOW);// depois de verificar a rotacao dos motores podemos alterara esta variavel
    U1 = U1 * -1;
  }
  else if(U2 < 0 && U1>0 ){
    digitalWrite(DIR1, LOW); //Se erro por positivo os motores andam para o outro lado
    digitalWrite(DIR2, HIGH);
    U2 = U2 * -1;
  }
// ***************************************************************************************************
// **********************Colocamos os valores na PWM ***********************************
  analogWrite (PWM1, U1); //pwm1 motor direita vista por tras
  analogWrite (PWM2, U2);//pwm2 motor esquerda vista por tras
// ***************************************************************************************************
// ******************Verifica se recebemos alguma coisa por bluetooth*******************
  if (Serial1.available()) {
   byte_atual = Serial1.read();
 Serial.print("recebi=");
 Serial.println(byte_atual);
  if(byte_anterior == 88)//88 recebemos X//*************Se recebmos um X o proximo valor será o valor de X
    {
     valor_x = byte_atual;
    }
  if(byte_anterior == 89)//89 recebemos Y//*************Se recebmos um Y o proximo valor será o valor de Y
    {
     valor_y = byte_atual;
   }
   if((byte_anterior != 88)||(byte_anterior != 89)) byte_anterior = byte_atual;// SE nao for X nem Y actualiza

  flag_joy=1;
  joystick();
 }
}
```

```
void read_mpu_6050_data() {                             //Subroutine for reading the raw gyro and accelerometer data
  Wire.beginTransmission(0x68);                         //Start communicating with the MPU-6050
  Wire.write(0x3B);                                  //Send the requested starting register
  Wire.endTransmission();                               //End the transmission
  Wire.requestFrom(0x68, 14);                           //Request 14 bytes from the MPU-6050
  while (Wire.available() < 14);                        //Wait until all the bytes are received
  acc_x = Wire.read() << 8 | Wire.read();                 //Add the low and high byte to the acc_x variable
  acc_y = Wire.read() << 8 | Wire.read();                 //Add the low and high byte to the acc_y variable
  acc_z = Wire.read() << 8 | Wire.read();                 //Add the low and high byte to the acc_z variable
  temperature = Wire.read() << 8 | Wire.read();             //Add the low and high byte to the temperature variable
  gyro_x = Wire.read() << 8 | Wire.read();                //Add the low and high byte to the gyro_x variable
  gyro_y = Wire.read() << 8 | Wire.read();                //Add the low and high byte to the gyro_y variable
  gyro_z = Wire.read() << 8 | Wire.read();                //Add the low and high byte to the gyro_z variable

}

void setup_mpu_6050_registers() {
  //Activate the MPU-6050
  Wire.beginTransmission(0x68);                         //Start communicating with the MPU-6050
  Wire.write(0x6B);                                  //Send the requested starting register
  Wire.write(0x00);                                  //Set the requested starting register
  Wire.endTransmission();                               //End the transmission
  //Configure the accelerometer (+/-8g)
  Wire.beginTransmission(0x68);                         //Start communicating with the MPU-6050
  Wire.write(0x1C);                                  //Send the requested starting register
  Wire.write(0x10);                                  //Set the requested starting register
  Wire.endTransmission();                               //End the transmission
  //Configure the gyro (500dps full scale)
  Wire.beginTransmission(0x68);                         //Start communicating with the MPU-6050
  Wire.write(0x1B);                                  //Send the requested starting register
  Wire.write(0x08);                                  //Set the requested starting register
  Wire.endTransmission();                               //End the transmission
}

void joystick() {
//*********************************************
//funcao que determina a direcao que o utilizador quer para o veiculo
if(valor_x==100||valor_x==0)// nestes valores o X é 0
{
  velocidade_x=0;
}
if(valor_y==100||valor_y==0)////nestes valores o y e 0
{
velocidade_y=0;
}

//*******************************************
//Diferentes valores correspondem a diferentes direcoes
//a baixo esta o mapa dos valores para as direcoes
if(valor_x>100)//andar direita
velocidade_x=(valor_x-100)*ganho_velocidade_app_x;//eSTA FUncao regulava a velocidade pretendida pelo utilizador alem da
direcaoif(valor_x>0&&valor_x<100)//esquerda
//velocidade_x=99;
if(valor_x>0&&valor_x<100)//esquerda
velocidade_x=(valor_x-100)*ganho_velocidade_app_x;
//velocidade_x=-99;
if(valor_y>0&&valor_y<100)//andar (frente/cima)
velocidade_y=(valor_y-100)*(-ganho_velocidade_app_y);
//velocidade_y=99;
if(valor_y>100)//andar (tras/baixo)
velocidade_y=(valor_y-100)*(-ganho_velocidade_app_y);
//velocidade_y=-99;
//Serial.println(velocidade_y);
}
```