Student: Moldovan Ovidiu
Instructor's Name: Roxana Both
Course: Control Engineering II
Group: 30331

mBot Nervous Bird

Table of contents

1. Project Description

An mBot kit is given, with the following key components: **Arduino ATmega2560**, **2 DC motors** with **encoders** and a **gyroscope**. The aim is to stabilise the robot and have it moving with a constant speed.

Since the connections are made through a provided interface (mCore) with RJ25 connectors, there are two possibilities for the control:

1. mBlock - Scratch 2.0 based, graphical programming IDE
2. the **MeAuriga** library provided for the Arduino IDE

After running examples on both IDEs, Arduino is chosen due to more flexibility offered by writing C code. First of all, a basic knowledge of Arduino programming is necessary. Also, reading the mBot manual, specifications and data sheet is needed in order to have an understanding of the mBot's different functionalities.

Due to the mCore interface, getting to know the MeAuriga library is mandatory, especially the following functions and methods: **setMotorPwm(int speed)**, **getCurrentSpeed()**, **gyro.getAngle(int axis)**.

2. Description and model of the two-wheeled inverted pendulum

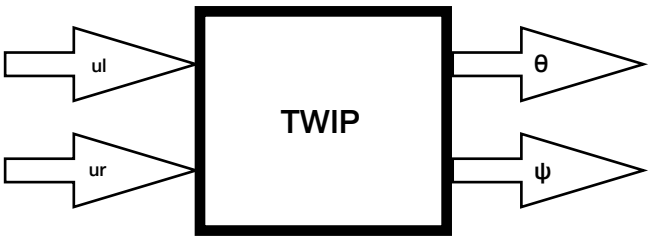

Figure 2.1

Figure 2.1 represents the MIMO system, where:

ul, ur = inputs to each wheel,

ψ = body pitch angle,

θ = angular speed,

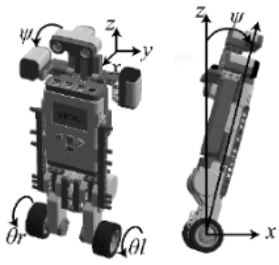

Figure 2.2

As exemplified in *Figure 2.2*.

The model in state space representation is: $\dot{x}_1 = A_1 x_1 + B_1 u$

where

$$x_1 = \begin{bmatrix} \theta \\ \psi \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \qquad u = \begin{bmatrix} u_l \\ u_r \end{bmatrix}$$

and

$$A_1 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & a_{32} & a_{33} & a_{34} \\ 0 & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

$$B_1 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ b_3 & b_3 \\ b_4 & b_4 \end{bmatrix}$$

By doing some measurements, checking the specification sheet (*Figure 2.4*) and using some approximations found in other papers based on this subject (example: *Figure 2.3*), I was able to model the system.

```
1.  M = 0.5; %mBot mass
2.  R = 0.03; %wheel radius
3.  W = 0.133; %body width
4.  D = 0.05; %body depth
5.  H = 0.129; %body height
6.  g = 9.8; %gravity acceleration
7.  L = H/2; %distance of the center of
    the mass from the wheel axle
```

Figure 2.3

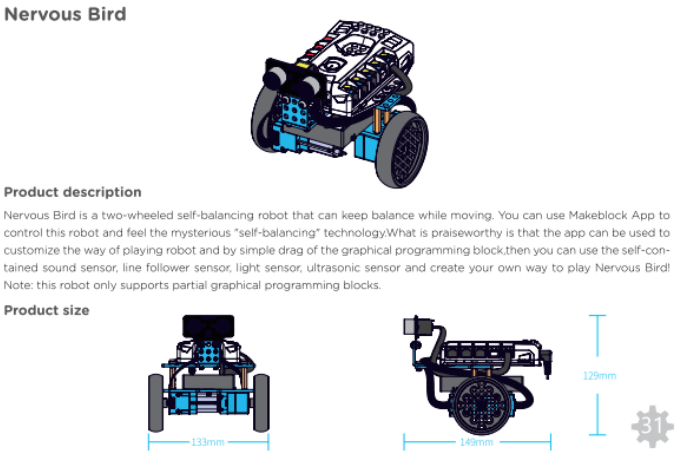

Figure 2.4

The A and B matrices are obtained as follows:

```
A =

         0         0    1.0000         0
         0         0         0    1.0000
         0 -466.8355 -284.3186  284.3186
         0  271.4790  113.8870 -113.8870
```

```
B =

         0         0
         0         0
  110.6927  110.6927
 -110.6927 -110.6927
```

After discretisation and getting the eigenvalues of the A matrix (full MATLAB code available in *Appendix 1*), the following poles are obtained: 1.0000, 0.0185, 1.0836, 0.9277. One pole on the unit circle and one outside of it evidence the instability of the open-loop system.

3. Identification

3.1. Speed control identification

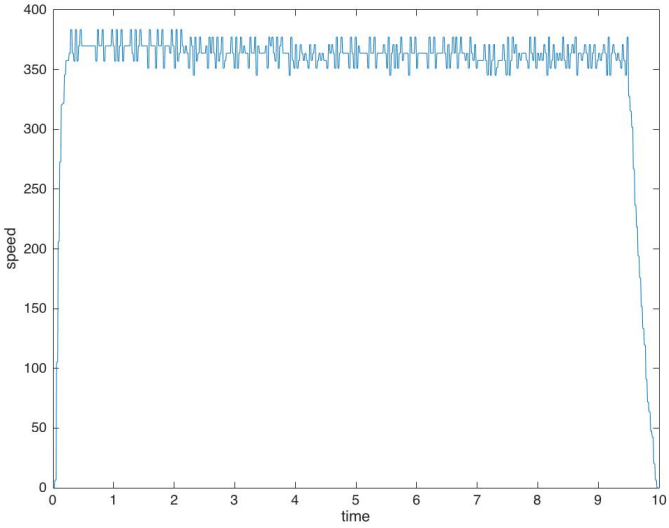

Figure 3.1

After doing a first order system identification (*Figure 3.1*) for the speed of the motor (MATLAB code in *Appendix 2*, Arduino code in *Appendix 3*), the following transfer function is obtained:

$$H(s) = \frac{1.428}{0.06914s + 1}$$

And the discretised one:

$$H(z) = \frac{0.2051}{z - 0.9856}$$

**Remark: An identification based on the encoder pulses was also tried, but was not further more pursued. The only notable result was that the available encoder has a precision of 1 degree (360 pulses / rotation).*

3.2. Gyroscope identification

For the stabilisation part, the input considered is the motors' speed and the output the measured 'z' axis of the gyroscope representing the body pitch angle of the mBot.

```
net = feedforwardnet([5 8 2]);
net.layers{1}.transferFcn = 'logsig';
net.layers{2}.transferFcn = 'radbas';
net.layers{3}.transferFcn = 'purelin';

NET = cascadeforwardnet(20);
N = neuralnet(NET);
n = neuralnet(net);
Model1 = nlarx(dat, [2 2 1], N);
Model2 = nlarx(dat, [2 2 1], n);
```

Figure 3.2

By using two of the examples given in MATLAB Documentation, a machine learning model was attempted (*Figure 3.2*), but the model obtained could not be linearised.

By using the System Identification Toolbox:

```
H = tfest(dat,3,1);
```

the following transfer function was obtained:

$$H(s) = \frac{10.83s + 107.7}{s^3 + 10.47s^2 + 141.2s + 88.74}$$

The MATLAB code can be found in *Appendix 4* and the Arduino code in *Appendix 5*.

4. Controller Design

After the identification, 2 different functionalities are being implemented. After the design, the two controllers are implemented as in the schematic in *Figure 4.*

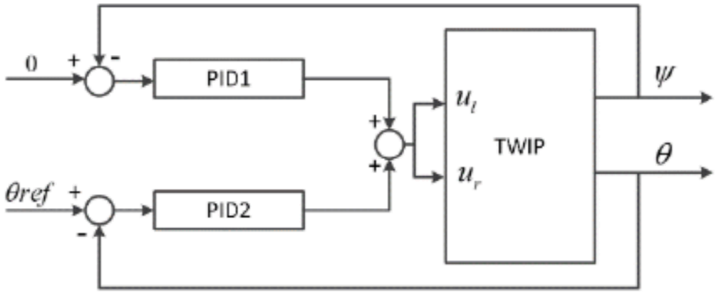

Figure 4

4.1. Stabilisation PID

4.1.1. Guillemin-Truxal method controller

This was the first attempt. After implementing it on the Arduino board, the results are very unsatisfying.

$$Hc(z) = \frac{26.48z^3 76.46z^2 + 73.83z - 23.85}{z^3 - 2.757z^2 + 2.528z - 0.7712}$$

4.1.2. Kessler method controller

Even though it is slow, it is definitely an improvement. With this controller the robot tries to stabilise itself and the speed of the motors change with the robot's incline, but the reaction speed seems to be a problem.

$$Hc(s) = \frac{s^3 + 10.47s^2 + 141.2s + 88.7}{21.66s^2 + 237.1s + 215.4}, \qquad Hc(z) = \frac{1.109z^3 - 1.843z^2 + 1.267z - 0.4807}{z^3 - 0.2406z^2 - 0.9367z + 0.3038}$$

4.1.3. Oscar Camacho's 'trial and error' controller

In two papers published by Oscar Camacho (see *References* section), a controller designed by 'trial and error' is presented for a similar project. Unfortunately, even though it works better than the Guillemin-Truxal one, the robot is still not stabilised.

	Variable	K_P	K_I	K_D
PID1	ψ	-77.97	-0.01	-8.79
PID2	θ	-1.07	-0.01	-1.36

4.2. Speed PID

Designed with the Kessler method, this controller does a good job maintaining a constant speed with no position error in steady state (*Figure 4.2*).

$$Hc(s) = \frac{0.06914s + 1}{0.0137s^2 + 0.1856s}$$

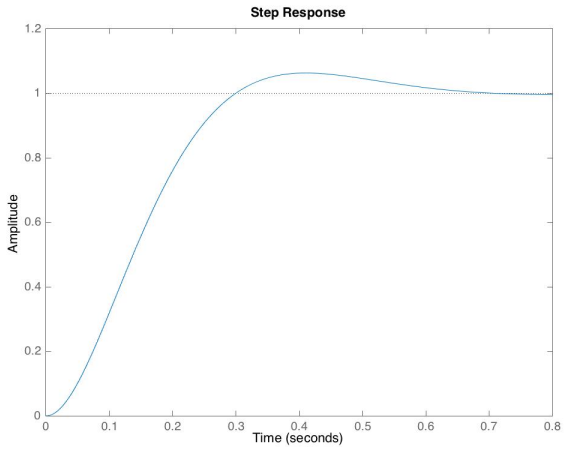

Figure 4.2

5. Arduino implementation

The following 2 variables are the command variables.

```
1.  float ref = 0;
2.  float com = 0;
```

Where *ref* is the angle at which the robot stabilises and *com* the speed given for the robot to move forward or backward.

Then, the feedback loop is computed as follows:

```
1.  float psi = gyro.getAngle(1);
2.  float theta = Encoder_1.getCurrentSpeed();
3.  e1[0] = ref - psi;
4.  e2[0] = com - Encoder_1.getCurrentSpeed();
```

And the two controllers are implemented in the following way:

```
5.  c1[0] = c1[1]*0.2406 + c1[2] * 0.9367 - c1[3] * 0.3038 + e1[0]*1.109 -
     e1[1]*1.843 +e1[2] * 1.267 - e1[3]*0.4807;
6.  c2[0] =c2[1]*1.873 - c2[2]*0.8731 + e2[0]*0.2534 + e2[1]*0.003418 -
     e2[2]*0.02192;
```

The full Arduino code can be found in *Appendix 6*.

6. References

1. JUAN VILLACRÉS, MICHELLE VISCAINO, MARCO HERRERA, OSCAR CAMACHO, Controllers Comparison to stabilize a Two-wheeled Inverted Pendulum: PID, LQR and Sliding Mode Control
2. Juan Villacrés, Michelle Viscaíno, Marco Herrera, Oscar Camacho, Real-Time Implementation of Different Controllers for a Two-wheeled Inverted Pendulum
3. M. Sami Fadali, Discrete-Time State-Space Equations
4. Cristina I. Pop, Eva H. Dulf, Clement Feştilă, Ingineria Reglării Automate 1, îndrumător de laborator
5. Makeblock, mBot documentation

7. Appendix

Appendix 1 - MATLAB Model of the Two Wheel Inverted Pendulum

```
m = 0.03; %wheel mass
M = 0.5; %mBot mass
R = 0.03; %wheel radius
Jw = m*(R^2)/2; %wheel inertia moment
n = 1; %gear ratio;
W = 0.133; %body width
D = 0.05; %body depth
H = 0.129; %body height
g = 9.8; %gravity acceleration
L = H/2; %distance of the center of the mass from the wheel axle
Jfork = M*(L^2)/3; %body pitch inertia moment
Jphi = M*((W^2) + (D^2))/12; %body yaw inertia moment
Jm = 1e-5; %dc motor inertia moment
Rm = 6.69; %[ohm], dc motor resistance
Kb = 0.468; % [V sec / rad] dc motor back emf constant
Kt = 0.317; %[Nm/A] dc motor torque constant
fm = 0.0022; %friction coefficient between body and dc motor
fw = 0; % friction coefficient between body and motion surface

e12 = M*L*R - 2*n^2*Jm;
e11 = (2*m + M) * R^2 + 2*Jw + 2*n^2*Jm;
e22 = M*(L^2) + Jfork + 2*n^2*Jm;

detE = e11*e22-(e12^2);

alpha = n*Kt/Rm;
beta = n*Kt*Kb/Rm + fm;
sigma = beta + fw;

a32 = -g*M*L*e12/detE;
a42 = g*M*L*e11/detE;
a33 = -2*(sigma*e22 + beta*e12) / detE;
a43 = 2 * (sigma * e12 + beta * e11)/detE;
a34 = 2 * beta * (e22 + e12) / detE;
a44 = -2 * beta *(e11 + e12) / detE;
b3 = alpha * (e11 + e12) / detE;
b4 = -alpha * (e11 + e12) / detE;

A = [0 0 1 0; 0 0 0 1; 0 a32 a33 a34; 0 a42 a43 a44];
B = [0 0; 0 0; b3 b3; b4 b4];

%Discretizing

Ad = expm(A*0.01);
Bd = A\(Ad - eye(size(Ad))) * B;
eig(A); %poles => unstable
```

Appendix 2 - MATLAB Speed transfer function identification

```
output
= [0.00 0.00 0.00 0.00 6.53 6.53 6.53 105.10 105.10 105.10 206.16 206.16 206
.
16 272.86 272.86 272.86 320.69 320.69 320.69321.99 321.99 321.99 345.62 345.
62 345.62 357.75 357.75 [..] −357.75 357.75 357.75 345.41 345.41 345.41 377.
28 377.28 377.28327.85 327.85 327.85 315.30 315.30 315.30 301.83 301.83 301.
83 266.79 266.79 266.79 236.48 236.48 236.48 218.29 218.29 218.29194.03 194.
03 194.03 175.84 175.84 175.84 151.59 151.59 151.59 133.40 133.40 133.40 119
.
47 119.47 119.47 90.95 90.95 90.95 71.8371.83 71.83 63.67 63.67 63.67 47.54
47.54 47.54 42.44 42.44 42.44 20.37 20.37 20.37 6.53 6.53 6.53 0.00 0.00 0.0
0 0.00];

time = 0:10/length(output):10−10/length(output);


plot(time,output)

input = ones(1, length(output)) * 255;

input(1:4)=0;

input(1100:length(input)) = 0;


T2 = time(13); %0.63 * yss = 229

T1 = time(5);

T = T2 − T1;

yss = mean(output(300:500));

y0 = 0;

uss = 255;

u0 = 0;

K = yss/uss;

H = tf([K], [T, 1])


figure()

Hz = c2d(H, 0.001, 'zoh')

Ho = feedback(Hz,1)

step(Ho)


Hc = tf([0.06914 1], [0.0137 0.1856 0]);

Hcz = c2d(Hc,0.001, 'tustin')

Ho = feedback(Hcz*Hz, 1);
step(Ho)
```

Appendix 3 - Arduino code for the speed transfer function identification

```
#include <Arduino.h>
#include <Wire.h>
#include <SoftwareSerial.h>
#include <MeAuriga.h>

MeEncoderOnBoard Encoder_1(SLOT1);
MeEncoderOnBoard Encoder_2(SLOT2);

void move(int direction, int speed)
{
      int leftSpeed = 0;
      int rightSpeed = 0;
      if(direction == 1){
            leftSpeed = -speed;
            rightSpeed = speed;
      }else if(direction == 2){
            leftSpeed = speed;
            rightSpeed = -speed;
      }else if(direction == 3){
            leftSpeed = -speed;
            rightSpeed = -speed;
      }else if(direction == 4){
            leftSpeed = speed;
            rightSpeed = speed;
      }
      Encoder_1.setTarPWM(leftSpeed);
      Encoder_2.setTarPWM(rightSpeed);
}


void setup() {
  //Set Pwm 8KHz
    TCCR1A = _BV(WGM10);
    TCCR1B = _BV(CS11) | _BV(WGM12);
    TCCR2A = _BV(WGM21) | _BV(WGM20);
    TCCR2B = _BV(CS21);
    attachInterrupt(Encoder_1.getIntNum(), isr_process_encoder1, RISING);
    attachInterrupt(Encoder_2.getIntNum(), isr_process_encoder2, RISING);
    Serial.begin(9600);
}
void loop(){
    move(1,255);
    _loop();
}
void _loop() {
  Encoder_1.loop();
  Encoder_2.loop();
  Serial.println(Encoder_1.getCurrentSpeed());
}
```

Appendix 4 - MATLAB Gyroscope transfer function identification

```
A = [1082        -3.17   34.61
1185        1.62    38.81
1289        3.49    35.91
1393        0.26    -29.38
1498        -13.26  -47.34
1601        -21.18  -49.45
1705        -18.38  34.61
1810        -0.55   52.24
1913        8.32    57.14
2018        14.24   53.87
2121        17.57   -29.10
2226        2.01    -54.39
[..]
14818       17.07   -24.72
14922       9.49    -45.27
15026       5.06    -51.10
15130       0.34    -53.87
15235       -7.27   -53.87
15338       -14.12  -56.04]
n = length(A(:,1));
time = A(:,1);
angle = A(:,2);
input = A(:,3);
plot(time,angle);xlabel('time');ylabel('degrees');
dat = iddata(angle,input,0.01);
%plot(dat)
H = tfest(dat,3,1)
H = tf(H);
```

Appendix 5 - Arduino Gyroscope transfer function identification

```
MeGyro gyro(1,0x69);

void setup() {
[...]
    gyro.begin();
}
void loop(){
  float m = gyro.getAngle(1);
  gyro.fast_update();
  long t = millis();
  Serial.print("\t");
  Serial.print(m);
  Serial.print("\t");
  delay(100); %0.1 sampling time

  Serial.print(t);
  Serial.print("\t");
if(m>0)
  Forward();
  if(m<0)
  Backward();
}
```

Appendix 6 - Arduino controller implementation

```
1.  float psi = gyro.getAngle(1);
2.  float theta = Encoder_1.getCurrentSpeed();
3.  //theta = gyro.getAngle(3) can be used if the identification is based on the
    gyroscope data, moving on x plane
4.  gyro.fast_update();
5.
6.  e1[0] = ref - psi;
7.  e2[0] = com - Encoder_1.getCurrentSpeed();
8.
9.  moveSpeed = 5*(c1[0]);
10.
11. c1[0] = c1[1]*0.2406 + c1[2] * 0.9367 - c1[3] * 0.3038 + e1[0]*1.109 -
     e1[1]*1.843 +e1[2] * 1.267 - e1[3]*0.4807;   c2[0] =c2[1]*1.873 -
     c2[2]*0.8731 + e2[0]*0.2534 + e2[1]*0.003418 - e2[2]*0.02192;
12.
13. for(int i = 3; i>0; i--){
14. c1[i] = c1[i-1];
15. e1[i] = e1[i-1];
16. e2[i] = e2[i-1];
17. }
18.
19. if(c1[0]<-255) c1[0] = -255;
20. if(c1[0]>255) c1[0] = 255;
21. if(c2[0]<-255) c2[0] = -255;
22. if(c2[0]>255) c2[0] = 255;
23. if(moveSpeed<-255) moveSpeed = -255;
24. if(moveSpeed>255) moveSpeed= 255;
25.
26. Serial.println(moveSpeed);
27. move(1,moveSpeed);
```