Individual Project

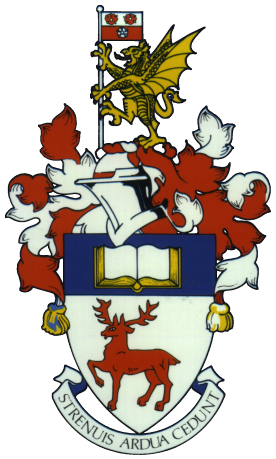
Pau Miquel Mir

28023668

Supervisor: Juanito Lopez

Word count: 187

May 2018



This report is submitted in partial fulfillment of the requirements for the MEng Mechanical Engineering, Faculty of Engineering and the Environment, University of Southampton.

Declaration

I, Pau Miquel Mir, declare that this thesis and the work presented in it are my own and has been generated by me as the result of my own original research. I confirm that:

1. This work was done wholly or mainly while in candidature for a degree at this University;
2. Where any part of this thesis has previously been submitted for any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. None of this work has been published before submission.

Acknowledgements

I want to thank my advisor for his time and dedication.

I also want to thank my parents, for their constant support and for proofreading the paper.

Abstract

This is the abstract.

Contents

Acronyms

ABS Anti-lock Braking System

SVM Support Vector Machine

1 Introduction

{sec:introduction}

1. Title page pretty

2. Word count glossaries

1.1 Proves

First use: Support Vector Machine (SVM). Second use: SVM.

First use: Anti-lock Braking System (ABS). Second use: ABS.

Prova [1]. Also, here is a trial on Fig. 1

This is a piece of shit. Fuck Prova merda I just keep on typing and want to see what happens, there is simply a bit of a lag and that it and that's it, shit this isn't working. It is, it is simply very very slow. I want to see if I can keep on typing and see what happens, it is quite nice indeed. How many words can I get without it working this is pretty cool and now I want to in insert a reference to Fig 1 This is interesting. What if I want to [1], and also [2], finally [3]

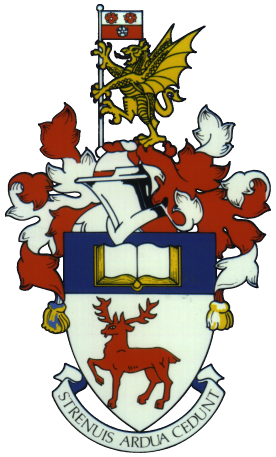


FIGURE 1: This caption is really really long and uses up more than one line to test the caption package to see what the fuck happens. This is a test. Caca.

{caca}

In Fig. 1 we can see a beautiful coat of arms. I am going to add eight more words. afjhffa

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.1}$$

{equ:prova}

1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

2 Conclusion

{sec:conclusion}

This is the conclusion. This is shit This is crap

Appendices

A Prova

{app:prova}

This is a new Appendix.

$$3+3=6 \tag{A.1}$$

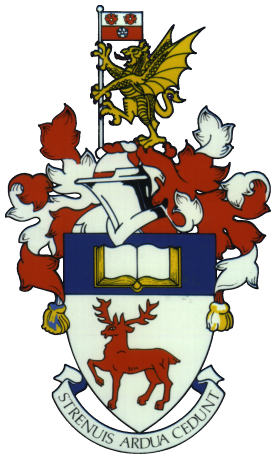


FIGURE A.1: Prova

B Eurobot Code

{app:placeholder}

```
/**************************************
Eurobot Project Spring 2017

Group 4 Space Ballerz

Group members:
 - Alberto Bosco
 - Mackenzie Brown
 - Michael Comport
 - Ian Hind Escolano
 - Pau Miquel Mir


Some MD25 code adapted from James Henderson's example code
Some servo code adapter from Scott Fitzgerald's example code
**************************************/


#include <SoftwareSerial.h>
#include <Wire.h>
#include <Servo.h>
#include <stdio.h>
#include <stdlib.h>

#define MD25ADDRESS         0x58    //Address of the MD25
#define CMD                 0x10    //Byte to 'write' to the CMD
#define SPEEDL              0x00    //Byte to send speed to first motor
```

```
28  #define SPEEDR              0x01     //Byte to send speed to second motor
29  #define ENCODER1            0x02     //Highest byte of motor encoder 1
30  #define ENCODER2            0x06     //Highest byte of motor encoder 2
31  #define RESETENCODERS       0x20     //Byte to reset encode registers to 0
32  #define ACCELERATION        0x0E     //Byte to send acceleration to the motors
33
34  Servo * Servos     = new Servo[3];
35  Servo * ServosTurn = new Servo[3];
36
37
38  ////////////////////////////////
39  // GLOBAL VARIABLE DEFINITIONS //
40  ////////////////////////////////
41
42
43  float   gWheelDiameter      = 100.0;
44  float   gWheelbase          = 280.0;
45  float   gDefaultSpeed       = 65.0;
46  int     gAcceleration       = 3;
47  int     gDeceleration       = 5;
48
49  // Distance in degrees the motor tends to run over at that speed
50  float   gBaseOffset         = 225.0;
51
52  // Distance in degrees the motor takes to reach the gDefaultSpeed at gAcceleration
53  float   gCutoff             = 480.0;
54
55  float   gCorrectionSpeed    = 5;
56
57  int     gCorrectionCounter  = 0;
58  int     gCorrectionMaximum  = 5;
59
60  int     gBackTrigPin        = 14;      //Back avoidance trigger pin
61  int     gBackEchoPin        = 15;      //Back avoidance echo pin
62  int     gSwitchPin          = 5;      //Track change pin
63  int     gPowerPin           = 6;      //Pullcord pin
64  int     gFrontTrigPin       = 7;      //Front avoidance trigger pin
65  int     gFrontEchoPin       = 8;      //Front avoidance echo pin
66  int     gRocketPin          = 10;      //Rocket launch pin
67
68  bool    gIsYellow            = true;
69
70  float   gDefaultDistanceLimit = 300.0;
71  unsigned long gStartTime;
72
73  int gDegreesToOpen = 140;
74  int gDegreesToTurn = 100;
75  int gDelayTime = 100;
76
77
78  ////////////////////////////////
79  //FUNCTION FORWARD DEFINITIONS //
80  ////////////////////////////////
81
82
83  void driveStraight(float distance, float speed = gDefaultSpeed,
84          float baseOffset = gBaseOffset, float cutoff = gCutoff,
85          int acceleration = gAcceleration, int deceleration = gDeceleration,
86          bool shouldAvoid = true);
87
88  void turnOnSpot(float degrees, float speed = gDefaultSpeed,
89        float baseOffset = gBaseOffset, float cutoff = gCutoff,
90        int acceleration = gAcceleration, int deceleration = gDeceleration,
91        bool shouldAvoid = true);
92
93  void driveWheels(float rightSpeed, float leftSpeed, float degrees,
```

```
94          float baseOffset = gBaseOffset, float cutoff = gCutoff,
95          int acceleration = gAcceleration, int deceleration = gDeceleration,
96          bool shouldAvoid = true);
97
98  void  stopMotor(int deceleration = gDeceleration);
99  long  encoder(int encoderNumber = 1);
100 void  encodeReset();
101 float encoderAverage();
102 void  sendByte(byte byteAddress, int value);
103 float distanceToDegrees(float distance);
104 float onspotDegreesToWheelDegrees(float degrees);
105 float distance(bool isForward = true);
106 bool  isClear(bool isForward = true, float distanceLimit = gDefaultDistanceLimit);
107 void  launchRocket();
108 bool  isTimeUp();
109
110 ////////////////////////////////
111 ////////////// SETUP //////////////
112 ////////////////////////////////
113
114 void setup(){
115   Serial.begin(9600);
116   Wire.begin();
117   sendByte(ACCELERATION, gAcceleration);
118   delay(200);
119   encodeReset();
120
121   pinMode(gFrontTrigPin, OUTPUT);     // Sets the trigPin as an Output
122   pinMode(gFrontEchoPin, INPUT);      // Sets the echoPin as an Input
123   pinMode(gBackTrigPin, OUTPUT);      // Sets the trigPin as an Output
124   pinMode(gBackEchoPin, INPUT);       // Sets the echoPin as an Input
125   pinMode(gSwitchPin, INPUT_PULLUP);  // Set Switch pin as an input
126   pinMode(gPowerPin, INPUT_PULLUP);   // Set Power pin as an input
127   pinMode(gRocketPin, OUTPUT);        // Set Rocket pin as an output
128
129
130   // SETUP FOR GRIPPER SERVOS
131   // Attach servos which open and close gripper arms to pins 11,12,13
132   Servos[0].attach(11);
133   Servos[1].attach(12);
134   Servos[2].attach(13);
135
136   // Attach servos which rotate gripper to pins 2,3,4
137   ServosTurn[0].attach(2);
138   ServosTurn[1].attach(3);
139   ServosTurn[2].attach(4);
140
141   // Initialise initial servo positions
142   Servos[0].write(5);
143   Servos[1].write(5);
144   Servos[2].write(5);
145   ServosTurn[0].write(22);
146   ServosTurn[1].write(85);
147   ServosTurn[2].write(10);
148
149   // Setup rocket to primed
150   digitalWrite(gRocketPin, LOW);
151
152
153   // Do nothing if the pullswitch hasn't been pulled.
154   while(digitalRead(gPowerPin) == LOW){ /* do nothing */ };
155
156   gStartTime = millis();
157
158   // Begin moving the robot
159   if (digitalRead(gSwitchPin) == HIGH) {
```

```
160       yellow();
161     }
162
163     else if(digitalRead(gSwitchPin) == LOW) {
164       blue();
165     }
166
167   }
168
169
170   ////////////////////////////////
171   ///////  Move Functions  ////////
172   ////////////////////////////////
173
174
175   void yellow() {
176
177     driveStraight(770, 24);
178     closeGripper(1);
179
180     // Picked up First cylnder
181
182     turnOnSpot(-87);
183     driveStraight(810, 45);
184     closeGripper(0);
185
186     // Picked up Second cylinder
187
188     turnOnSpot(-55);
189     delay(100);
190     driveStraight(250, 35);
191     closeGripper(2);
192     driveStraight(10,40);
193     delay(100);
194
195     // Picked up Third cylinder
196
197     turnOnSpot(-50,30);
198     turnGripperVertical(2);
199     turnGripperVertical(0);
200     turnGripperVertical(1);
201     delay(100);
202
203     // Go forward without corrections to gently hit the wall and turn straight.
204
205     unsigned long currentTime = millis();
206     while(millis() - currentTime < 1500){
207       sendByte(ACCELERATION, gAcceleration);
208       sendByte(SPEEDR, 128+30);
209       sendByte(SPEEDL, 128+30);
210     }
211
212     openGripper(0);
213     openGripper(1);
214     openGripper(2);
215
216     // Dropped 3 cylinders in side base
217
218     driveStraight(-320, 40);
219     turnGripperHorizontal(2);
220     turnGripperHorizontal(1);
221     turnGripperHorizontal(0);
222     turnOnSpot(-30);
223     driveStraight(285, 30);
224     closeGripper(0);
225
```

```
226   // Picked up Fourth cylinder
227
228   driveStraight(-180, 30);
229   turnOnSpot(155);
230   driveStraight(750 , 40);
231   turnOnSpot(10);
232   driveStraight(100,35);
233   closeGripper(2);
234   turnOnSpot(75);
235   turnGripperVertical(2);
236   turnGripperVertical(1);
237   turnGripperVertical(0);
238   driveStraight(50,20);
239   launchRocket();
240
241
242 }
243
244 void blue(){
245
246   driveStraight(748, 30);
247   closeGripper(1);
248
249   // Picked up First cylnder
250
251   turnOnSpot(87);
252   driveStraight(840, 45);
253   closeGripper(2);
254
255   // Picked up Second cylinder
256
257   turnOnSpot(55);
258   delay(300);
259   driveStraight(280, 35);
260   closeGripper(0);
261   driveStraight(10,40);
262   delay(200);
263
264   // Picked up Third cylinder
265
266   turnOnSpot(45,30);
267   turnGripperVertical(0);
268   turnGripperVertical(2);
269   turnGripperVertical(1);
270   delay(200);
271
272   // Go forward without corrections to gently hit the wall and turn straight.
273   unsigned long currentTime = millis();
274   while(millis() - currentTime < 1200){
275     sendByte(ACCELERATION, gAcceleration);
276     sendByte(SPEEDR, 128 + 30);
277     sendByte(SPEEDL, 128 + 30);
278   }
279
280   openGripper(0);
281   openGripper(1);
282   openGripper(2);
283
284   // Dropped 3 cylinders in side base
285
286   driveStraight(-319, 40);
287   turnGripperHorizontal(0);
288   turnGripperHorizontal(1);
289   turnGripperHorizontal(2);
290   turnOnSpot(30);
291   driveStraight(295, 30);
```

```
292   closeGripper(2);
293   driveStraight(10,40);
294
295   // Picked up Fourth cylinder
296   driveStraight(-190, 35);
297   turnOnSpot(-150);
298
299   driveStraight(700, 40);
300   turnOnSpot(-25);
301   driveStraight(250,30);
302   closeGripper(0);
303   closeGripper(1);
304   driveStraight(10,40);
305   turnGripperVertical(2);
306   turnGripperVertical(1);
307   turnGripperVertical(0);
308   driveStraight(50,30);
309   turnOnSpot(-80,30);
310   driveStraight(60,30);
311   openGripper(0);
312   openGripper(2);
313   launchRocket();
314 }
315
316
317 ////////////////////////////////
318 ///////// Drive Functions /////////
319 ////////////////////////////////
320
321
322 // Function to drive the wheels for a certain distance at certain speed. Speed
323 // should always be positive, a negative distance will make it go backwards.
324 void driveStraight(float distance, float speed, float baseOffset, float cutoff,
325                    int acceleration, int deceleration, bool shouldAvoid){
326
327   if (distance > 0){
328     driveWheels(speed, speed, distanceToDegrees(distance),
329                 baseOffset, cutoff, acceleration, deceleration, shouldAvoid);
330   }
331
332   if (distance < 0){
333     driveWheels(-speed, -speed, distanceToDegrees(distance*-1),
334                 baseOffset, cutoff, acceleration, deceleration, shouldAvoid);
335   }
336
337 }
338
339
340 // Function to make the robot spin in place by a certain amount of degrees,
341 // a positive angle will make the robot spin clockwise.
342 void turnOnSpot(float degrees, float speed, float baseOffset, float cutoff,
343                 int acceleration, int deceleration, bool shouldAvoid){
344
345   if (degrees > 0){
346     driveWheels(speed*-1, speed, onspotDegreesToWheelDegrees(degrees));
347   }
348   if (degrees < 0){
349     driveWheels(speed,speed*-1,onspotDegreesToWheelDegrees(degrees*-1));
350   }
351
352 }
353
354
355 //
356 // Internal function that drives the wheels. It takes as input the speed for the
357 // right and left motor, and the degrees of rotation. This value should be the
```

```
358 // average of the absolute value of the degrees each wheel will spin. It also takes
359 // a value of baseOffset, which is the degrees the motor tends to go over at that
360 // speed, and the cutoff, which is the degrees it takes to spin up to maximum speed.
361 //
362 void driveWheels(float rightSpeed, float leftSpeed, float degrees,
363                  float baseOffset, float cutoff, int acceleration,
364                  int deceleration, bool shouldAvoid){
365
366   bool isForward = true;
367   if(rightSpeed < 0 && leftSpeed < 0){
368     isForward = false;
369   }
370   // Calculate the deceleration time, using the data from the datasheet.
371   // Take in the values of the offset
372   float baseDecelerationTime = (abs(rightSpeed) + abs(leftSpeed)) /
373                                      float(gDeceleration) * 15;
374   float offset = baseOffset;
375   float decelerationTime = baseDecelerationTime;
376
377   // If the distance is less then the cutoff, that means the wheels are still
378   // accelerating. Therefore, the offset and the deceleration time must be
379   // adjusted accordingly, by a factor of the distance / cutoff distance
380   if (degrees <= cutoff){
381     offset = baseOffset * degrees/cutoff;
382     decelerationTime = baseDecelerationTime * degrees/cutoff;
383   }
384
385   // Reset the enconders
386   encodeReset();
387
388   // Wait for the encoder average to be larger than the degree value minus the
389   // offset, then stop the motor. Wait for the robot to come to a stop
390   while(encoderAverage() < abs((degrees-offset))){
391     if(is90secDone()) {
392       stopMotor(deceleration);
393       delay(decelerationTime);
394       launchRocket();
395       return;
396     }
397     if(!isClear(isForward)){
398       stopMotor();
399     }
400     else{
401       sendByte(ACCELERATION, acceleration);
402       sendByte(SPEEDR, 128 + rightSpeed);
403       sendByte(SPEEDL, 128 + leftSpeed);
404     }
405   }
406   stopMotor(deceleration);
407   delay(decelerationTime);
408
409   if (gCorrectionCounter < gCorrectionMaximum){
410     // Calculate new speeds that will be used for correction. First find the
411     // square root of the ratio between the rightSpeed and the leftSpeed. Then,
412     // multiply and divide gCorrectionSpeed by the rootSpeedRatio, thus
413     // achieving two speeds with the same ratio as the rightSpeed and leftSpeed,
414     // but centrered around gCorrectionSpeed. Then apply the same signs as the
415     // original rightSpeed and leftSpeed
416
417     gCorrectionCounter++;
418
419     float rootSpeedRatio = sqrt(abs(rightSpeed / leftSpeed));
420     float newRightSpeed;
421     float newLeftSpeed;
422
423     if (rightSpeed > 0){
```

```
424         newRightSpeed = gCorrectionSpeed * rootSpeedRatio;
425       }
426       else if (rightSpeed < 0){
427         newRightSpeed = -1 * gCorrectionSpeed * rootSpeedRatio;
428       }
429       if (leftSpeed > 0){
430         newLeftSpeed = gCorrectionSpeed / rootSpeedRatio;
431       }
432       else if (leftSpeed < 0){
433         newLeftSpeed = -1 * gCorrectionSpeed / rootSpeedRatio;
434       }
435
436       // If the distance is found to be different than the wanted by more than a
437       // degree, call the driveWheels function with new speeds.
438       if (encoderAverage() > degrees + 1.0){
439         driveWheels(-1 * newRightSpeed, -1 * newLeftSpeed, encoderAverage() - degrees,
440                     baseOffset / 10.0, cutoff / 10.0);
441       }
442       else if (encoderAverage() < degrees - 1.0){
443         driveWheels(newRightSpeed, newLeftSpeed, degrees - encoderAverage(),
444                     baseOffset / 10.0, cutoff/ 10.0);
445       }
446     }
447     gCorrectionCounter = 0;
448   }
449
450
451   // Function to stop motors.
452   void stopMotor(int deceleration){
453     sendByte(ACCELERATION, deceleration);
454     sendByte(SPEEDR, 128);
455     sendByte(SPEEDL, 128);
456   }
457
458
459   ////////////////////////////////
460   /////// Encoder Functions ///////
461   ////////////////////////////////
462
463
464   // Function to read and return the  value of an encoder as
465   // a long, takes the number of the encoder as an input.
466   long encoder(int encoderNumber){
467
468     Wire.beginTransmission(MD25ADDRESS);
469
470     if (encoderNumber == 1){
471       Wire.write(ENCODER1);
472     }
473
474     if (encoderNumber == 2){
475       Wire.write(ENCODER2);
476     }
477
478     Wire.endTransmission();
479     Wire.requestFrom(MD25ADDRESS, 4);      // Request 4 bytes from MD25
480
481     // Wait for 4 bytes to become available
482     while(Wire.available() < 4) { /* do nothing */};
483
484     long encoderValue = Wire.read();       // First byte for encoder 2, HH
485
486     for (int i = 0; i < 3; i++){           //
487       encoderValue <<= 8;                  // Read the next three bytes
488       encoderValue += Wire.read();         //
489     }
```

```
490
491   return(encoderValue);                  //Return encoderValue
492 }
493
494
495 // Function that returns the absolute value of the average of the two encoders
496 float encoderAverage(){
497   return( ( abs(encoder(1)) + abs(encoder(2)) ) / 2 );
498 }
499
500
501 // Function to set the encoder values to 0
502 void encodeReset(){
503   sendByte(CMD,RESETENCODERS);
504 }
505
506
507 ////////////////////////////////
508 /////// Gripper Functions ///////
509 ////////////////////////////////
510
511
512 void closeGripper(int servoNumber) {
513
514   if(is90secDone()) {
515     launchRocket();
516     return;
517   }
518
519   for (int pos = 5; pos <= gDegreesToOpen; pos += 1) {
520     Servos[servoNumber].write(pos);
521     delay(15);
522   }
523
524   delay(gDelayTime);
525
526 }
527
528
529 void openGripper(int servoNumber) {
530
531   if(is90secDone()) {
532       launchRocket();
533       return;
534   }
535
536   for (int pos = gDegreesToOpen; pos >= 5; pos -= 1) {
537     Servos[servoNumber].write(pos);
538     delay(15);
539   }
540
541   delay(gDelayTime);
542
543 }
544
545
546 void turnGripperVertical(int servoNumber) {
547
548   if(is90secDone()) {
549       launchRocket();
550       return;
551   }
552
553   //SERVO NO. 0
554   if(servoNumber == 0) {
555     for(int pos = 22; pos <= gDegreesToTurn; pos += 1) {
```

```
556         ServosTurn[servoNumber].write(pos);
557         delay(10);
558       }
559     }
560
561     //SERVO NO. 1
562     else if(servoNumber == 1) {
563       ServosTurn[servoNumber].write(5);
564     }
565
566     //SERVO NO. 2
567     else if(servoNumber == 2) {
568       for(int pos = 5; pos <= 95; pos += 1) {
569         ServosTurn[servoNumber].write(pos);
570         delay(5);
571       }
572     }
573
574     delay(gDelayTime);
575
576   }
577
578
579   void turnGripperHorizontal(int servoNumber) {
580
581     if(is90secDone()) {
582         launchRocket();
583         return;
584       }
585
586     //SERVO NO. 0
587     if(servoNumber == 0) {
588       ServosTurn[servoNumber].write(22);
589     }
590
591     //SERVO NO. 1
592     else if(servoNumber == 1) {
593       ServosTurn[servoNumber].write(85);
594     }
595
596     //SERVO NO. 2
597     else {
598       ServosTurn[servoNumber].write(10);
599     }
600
601     delay(gDelayTime);
602
603   }
604
605
606   ////////////////////////////////
607   ////// Obstacle Aboidance ///////
608   ////////////////////////////////
609
610
611   // Returns the distance away from either the front sensor when
612   // isForward = true, and the back sensor when isForward = false
613   float distance(bool isForward){
614
615     float duration;
616     float distance;
617
618     if(isForward){
619       digitalWrite(gFrontTrigPin, LOW);
620       delayMicroseconds(2);
621       // Sets the gTrigPin on HIGH state for 10 micro seconds
```

```
622       digitalWrite(gFrontTrigPin, HIGH);
623       delayMicroseconds(10);
624       digitalWrite(gFrontTrigPin, LOW);
625       // Reads the echoPin, returns the sound wave travel time in microseconds
626       duration = pulseIn(gFrontEchoPin, HIGH);
627       }
628
629     else{
630       digitalWrite(gBackTrigPin, LOW);
631       delayMicroseconds(2);
632       // Sets the gTrigPin on HIGH state for 10 micro seconds
633       digitalWrite(gBackTrigPin, HIGH);
634       delayMicroseconds(10);
635       digitalWrite(gBackTrigPin, LOW);
636       // Reads the echoPin, returns the sound wave travel time in microseconds
637       duration = pulseIn(gBackEchoPin, HIGH);
638       }
639
640     // Calculating the distance
641     distance = duration * 0.34 / 2;
642
643     return(distance);
644
645   }
646
647
648   // Returns true if the robot is further away than distanceLimit from the
649   // either the front when isForward = true and the back when isForward = false
650   bool isClear(bool isForward, float distanceLimit){
651     float currentDistance = distance(isForward);
652     return(currentDistance > distanceLimit);
653   }
654
655
656   ////////////////////////////////
657   /////// Rocket Functions ////////
658   ////////////////////////////////
659
660
661   // Stops motor and returns true if 90 second have pased since the
662   // pull cord was pulled, returns false otherwise.
663   bool is90secDone(){
664     if (millis() - gStartTime >= 90000){
665       stopMotor(gDeceleration);
666       delay(1000);
667       return true;
668     }
669     else {
670       return false;
671     }
672   }
673
674
675   // Waits until 90 seconds have transcurred since the pull cord was
676   // launched, then launches the rocket by pulsing gRocketPin
677   void launchRocket(){
678
679     while(is90secDone() == false){ /* Do nothing */}
680     delay(1000);
681     digitalWrite(gRocketPin, HIGH);
682     delay(1000);
683     digitalWrite(gRocketPin, LOW);
684     exit(0);
685
686   }
687
```

```
688
689 ////////////////////////////////
690 /////// Helper Functions ////////
691 ////////////////////////////////
692
693
694 // Function that sends a value to a byte address
695 void sendByte( byte byteAddress, int value ){
696   Wire.beginTransmission(MD25ADDRESS);
697   Wire.write(byteAddress);               //'Write' to byteaddress
698   Wire.write(value);                     //Send a value to that adress
699   Wire.endTransmission();
700 }
701
702
703 // Function to convert a distance to a degree value
704 float distanceToDegrees(float distance){
705   return(distance / gWheelDiameter / 3.1415 * 360);
706 }
707
708
709 // Function to find the degrees the wheels have to spin
710 // for a degree value of spinning on the spot
711 float onspotDegreesToWheelDegrees(float degrees){
712   return(distanceToDegrees(degrees / 360 * 3.1415 * gWheelbase));
713 }
714
715
716 // No need for a loop function
717 void loop(){
718 }
```

References

[1] J. Swift, *Gulliver's Travels*, ser. Diamond classics. Jones & Company, 1826. [Online]. Available: https://books.google.co.uk/books?id=ta1uaL7RF5gC.

[2] P. W. Higgs, "Broken symmetries and the masses of gauge bosons", *Physical review letters*, vol. 13, no. 16, pp. 508–509, Oct. 1964, ISSN: 00319007. DOI: 10.1103/PhysRevLett.13.508. arXiv: PhysRevLett.13.508 [10.1103]. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.13.508.

[3] J. A. Nelder and R. Mead, "A Simplex Method for Function Minimization", *The computer journal*, vol. 7, no. 4, pp. 308–313, Jan. 1965. DOI: 10.1093/comjnl/7.4.308. [Online]. Available: https://academic.oup.com/comjnl/article-lookup/doi/10.1093/comjnl/7.4.308.