Individual Project

Word count: 45

Pau Miquel Mir
28023668

May 2018

Abstract

This is the abstract.

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.

Contents

1 Introduction

These are four words. These are four more. And another 3. Two more. And now 4 more. This is come more stuff [1]. Mike is a bit of a moron, and he likes this book [2]. This can be seen in Fig. 2.

This is a new paragraph.

$$3+3=6 \tag{1.1}$$

FIGURE 1: Some coins

2 Conclusion

2.1 Future Work

FIGURE 2: Some coins

References

[1] E.M.L. Beale. Cycling in the dual simplex algorithm. *Naval Research Logistics Quarterly*, 2 (4):269–275, 1955. ISSN 1931-9193. doi: 10.1002/nav.3800020406.

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

Additional Reading

S. Dasgupta, C. Papadimitriou, and U. Vazirani. *Algorithms*. McGraw-Hill Education, 2008. ISBN 9780073523408. URL `http://books.google.com/books?id=LaIqnwEACAAJ`.

Florian A. Potra. Interior point methods, twenty years after. Lecture, September 2003. URL `http://www.math.umbc.edu/~potra/talk0930.pdf`.

Appendices

A 1 Appendix 1

```
/***************************************
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
#define SPEEDR              0x01     //Byte to send speed to second motor
#define ENCODER1            0x02     //Highest byte of motor encoder 1
#define ENCODER2            0x06     //Highest byte of motor encoder 2
#define RESETENCODERS       0x20     //Byte to reset encode registers to 0
#define ACCELERATION        0x0E     //Byte to send acceleration to the motors

Servo * Servos     = new Servo[3];
Servo * ServosTurn = new Servo[3];


/////////////////////////////////
// GLOBAL VARIABLE DEFINITIONS //
/////////////////////////////////


float   gWheelDiameter      = 100.0;
float   gWheelbase          = 280.0;
float   gDefaultSpeed       = 65.0;
int     gAcceleration       = 3;
int     gDeceleration       = 5;

// Distance in degrees the motor tends to run over at that speed
float   gBaseOffset         = 225.0;

// Distance in degrees the motor takes to reach the gDefaultSpeed at gAcceleration
float   gCutoff             = 480.0;

float   gCorrectionSpeed    = 5;
```

```

int     gCorrectionCounter  = 0;
int     gCorrectionMaximum  = 5;

int     gBackTrigPin        = 14;      //Back avoidance trigger pin
int     gBackEchoPin        = 15;      //Back avoidance echo pin
int     gSwitchPin          = 5;      //Track change pin
int     gPowerPin           = 6;      //Pullcord pin
int     gFrontTrigPin       = 7;      //Front avoidance trigger pin
int     gFrontEchoPin       = 8;      //Front avoidance echo pin
int     gRocketPin          = 10;      //Rocket launch pin

bool    gIsYellow            = true;

float   gDefaultDistanceLimit = 300.0;
unsigned long gStartTime;

int gDegreesToOpen = 140;
int gDegreesToTurn = 100;
int gDelayTime = 100;


/////////////////////////////////
//FUNCTION FORWARD DEFINITIONS //
/////////////////////////////////


void driveStraight(float distance, float speed = gDefaultSpeed,
        float baseOffset = gBaseOffset, float cutoff = gCutoff,
        int acceleration = gAcceleration, int deceleration = gDeceleration,
        bool shouldAvoid = true);

void turnOnSpot(float degrees, float speed = gDefaultSpeed,
      float baseOffset = gBaseOffset, float cutoff = gCutoff,
      int acceleration = gAcceleration, int deceleration = gDeceleration,
      bool shouldAvoid = true);

void driveWheels(float rightSpeed, float leftSpeed, float degrees,
       float baseOffset = gBaseOffset, float cutoff = gCutoff,
       int acceleration = gAcceleration, int deceleration = gDeceleration,
       bool shouldAvoid = true);

void  stopMotor(int deceleration = gDeceleration);
long  encoder(int encoderNumber = 1);
void  encodeReset();
float encoderAverage();
void  sendByte(byte byteAddress, int value);
float distanceToDegrees(float distance);
float onspotDegreesToWheelDegrees(float degrees);
float distance(bool isForward = true);
bool  isClear(bool isForward = true, float distanceLimit = gDefaultDistanceLimit);
void  launchRocket();
bool  isTimeUp();

////////////////////////////////
//////////// SETUP /////////////
////////////////////////////////

void setup(){
  Serial.begin(9600);
  Wire.begin();
  sendByte(ACCELERATION, gAcceleration);
  delay(200);
  encodeReset();

  pinMode(gFrontTrigPin, OUTPUT);     // Sets the trigPin as an Output
  pinMode(gFrontEchoPin, INPUT);      // Sets the echoPin as an Input
  pinMode(gBackTrigPin, OUTPUT);      // Sets the trigPin as an Output
  pinMode(gBackEchoPin, INPUT);       // Sets the echoPin as an Input
  pinMode(gSwitchPin, INPUT_PULLUP);  // Set Switch pin as an input
  pinMode(gPowerPin, INPUT_PULLUP);   // Set Power pin as an input
  pinMode(gRocketPin, OUTPUT);        // Set Rocket pin as an output


  // SETUP FOR GRIPPER SERVOS
  // Attach servos which open and close gripper arms to pins 11,12,13
  Servos[0].attach(11);
  Servos[1].attach(12);
  Servos[2].attach(13);

  // Attach servos which rotate gripper to pins 2,3,4
  ServosTurn[0].attach(2);
  ServosTurn[1].attach(3);
  ServosTurn[2].attach(4);

  // Initialise initial servo positions
  Servos[0].write(5);
  Servos[1].write(5);
  Servos[2].write(5);
  ServosTurn[0].write(22);
  ServosTurn[1].write(85);
  ServosTurn[2].write(10);

  // Setup rocket to primed
  digitalWrite(gRocketPin, LOW);


  // Do nothing if the pullswitch hasn't been pulled.
  while(digitalRead(gPowerPin) == LOW){ /* do nothing */ };

  gStartTime = millis();

  // Begin moving the robot
  if (digitalRead(gSwitchPin) == HIGH) {
    yellow();
  }

  else if(digitalRead(gSwitchPin) == LOW) {
    blue();
  }

}


/////////////////////////////////
///////  Move Functions  ////////
/////////////////////////////////

```

```

void yellow() {

  driveStraight(770, 24);
  closeGripper(1);

  // Picked up First cylnder

  turnOnSpot(-87);
  driveStraight(810, 45);
  closeGripper(0);

  // Picked up Second cylinder

  turnOnSpot(-55);
  delay(100);
  driveStraight(250, 35);
  closeGripper(2);
  driveStraight(10,40);
  delay(100);

  // Picked up Third cylinder

  turnOnSpot(-50,30);
  turnGripperVertical(2);
  turnGripperVertical(0);
  turnGripperVertical(1);
  delay(100);

  // Go forward without corrections to gently hit the wall and turn straight.

  unsigned long currentTime = millis();
  while(millis() - currentTime < 1500){
    sendByte(ACCELERATION, gAcceleration);
    sendByte(SPEEDR, 128+30);
    sendByte(SPEEDL, 128+30);
  }

  openGripper(0);
  openGripper(1);
  openGripper(2);

  // Dropped 3 cylinders in side base

  driveStraight(-320, 40);
  turnGripperHorizontal(2);
  turnGripperHorizontal(1);
  turnGripperHorizontal(0);
  turnOnSpot(-30);
  driveStraight(285, 30);
  closeGripper(0);

  // Picked up Fourth cylinder

  driveStraight(-180, 30);
  turnOnSpot(155);
  driveStraight(750 , 40);
  turnOnSpot(10);
  driveStraight(100,35);
  closeGripper(2);
  turnOnSpot(75);
  turnGripperVertical(2);
  turnGripperVertical(1);
  turnGripperVertical(0);
  driveStraight(50,20);
  launchRocket();


}

void blue(){

  driveStraight(748, 30);
  closeGripper(1);

  // Picked up First cylnder

  turnOnSpot(87);
  driveStraight(840, 45);
  closeGripper(2);

  // Picked up Second cylinder

  turnOnSpot(55);
  delay(300);
  driveStraight(280, 35);
  closeGripper(0);
  driveStraight(10,40);
  delay(200);

  // Picked up Third cylinder

  turnOnSpot(45,30);
  turnGripperVertical(0);
  turnGripperVertical(2);
  turnGripperVertical(1);
  delay(200);

  // Go forward without corrections to gently hit the wall and turn straight.
  unsigned long currentTime = millis();
  while(millis() - currentTime < 1200){
    sendByte(ACCELERATION, gAcceleration);
    sendByte(SPEEDR, 128 + 30);
    sendByte(SPEEDL, 128 + 30);
  }

  openGripper(0);
  openGripper(1);
  openGripper(2);

  // Dropped 3 cylinders in side base

  driveStraight(-319, 40);
  turnGripperHorizontal(0);
  turnGripperHorizontal(1);
  turnGripperHorizontal(2);
  turnOnSpot(30);
  driveStraight(295, 30);
  closeGripper(2);
  driveStraight(10,40);

  // Picked up Fourth cylinder
  driveStraight(-190, 35);
  turnOnSpot(-150);

  driveStraight(700, 40);
  turnOnSpot(-25);
  driveStraight(250,30);
  closeGripper(0);
  closeGripper(1);
  driveStraight(10,40);
  turnGripperVertical(2);
  turnGripperVertical(1);
  turnGripperVertical(0);
  driveStraight(50,30);
  turnOnSpot(-80,30);
  driveStraight(60,30);
  openGripper(0);
  openGripper(2);
  launchRocket();
}


///////////////////////////////////
///////// Drive Functions /////////
///////////////////////////////////


// Function to drive the wheels for a certain distance at certain speed. Speed
// should always be positive, a negative distance will make it go backwards.
void driveStraight(float distance, float speed, float baseOffset, float cutoff,
                   int acceleration, int deceleration, bool shouldAvoid){

  if (distance > 0){
    driveWheels(speed, speed, distanceToDegrees(distance),
                baseOffset, cutoff, acceleration, deceleration, shouldAvoid);
  }

  if (distance < 0){
    driveWheels(-speed, -speed, distanceToDegrees(distance*-1),
                baseOffset, cutoff, acceleration, deceleration, shouldAvoid);
  }

}


// Function to make the robot spin in place by a certain amount of degrees,
// a positive angle will make the robot spin clockwise.
void turnOnSpot(float degrees, float speed, float baseOffset, float cutoff,
                int acceleration, int deceleration, bool shouldAvoid){

  if (degrees > 0){
    driveWheels(speed*-1, speed, onspotDegreesToWheelDegrees(degrees));
  }
  if (degrees < 0){
    driveWheels(speed,speed*-1,onspotDegreesToWheelDegrees(degrees*-1));
  }
```

```

}


//
// Internal function that drives the wheels. It takes as input the speed for the
// right and left motor, and the degrees of rotation. This value should be the
// average of the absolute value of the degrees each wheel will spin. It also takes
// a value of baseOffset, which is the degrees the motor tends to go over at that
// speed, and the cutoff, which is the degrees it takes to spin up to maximum speed.
//
void driveWheels(float rightSpeed, float leftSpeed, float degrees,
                 float baseOffset, float cutoff, int acceleration,
                 int deceleration, bool shouldAvoid){

  bool isForward = true;
  if(rightSpeed < 0 && leftSpeed < 0){
    isForward = false;
  }
  // Calculate the deceleration time, using the data from the datasheet.
  // Take in the values of the offset
  float baseDecelerationTime = (abs(rightSpeed) + abs(leftSpeed)) /
                                     float(gDeceleration) * 15;
  float offset = baseOffset;
  float decelerationTime = baseDecelerationTime;

  // If the distance is less then the cutoff, that means the wheels are still
  // accelerating. Therefore, the offset and the deceleration time must be
  // adjusted accordingly, by a factor of the distance / cutoff distance
  if (degrees <= cutoff){
    offset = baseOffset * degrees/cutoff;
    decelerationTime = baseDecelerationTime * degrees/cutoff;
  }

  // Reset the enconders
  encodeReset();

  // Wait for the encoder average to be larger than the degree value minus the
  // offset, then stop the motor. Wait for the robot to come to a stop
  while(encoderAverage() < abs((degrees-offset))){
    if(is90secDone()) {
      stopMotor(deceleration);
      delay(decelerationTime);
      launchRocket();
      return;
    }
    if(!isClear(isForward)){
      stopMotor();
    }
    else{
      sendByte(ACCELERATION, acceleration);
      sendByte(SPEEDR, 128 + rightSpeed);
      sendByte(SPEEDL, 128 + leftSpeed);
    }
  }
  stopMotor(deceleration);
  delay(decelerationTime);

  if (gCorrectionCounter < gCorrectionMaximum){
    // Calculate new speeds that will be used for correction. First find the
    // square root of the ratio between the rightSpeed and the leftSpeed. Then,
    // multiply and divide gCorrectionSpeed by the rootSpeedRatio, thus
    // achieving two speeds with the same ratio as the rightSpeed and leftSpeed,
    // but centred around gCorrectionSpeed. Then apply the same signs as the
    // original rightSpeed and leftSpeed

    gCorrectionCounter++;

    float rootSpeedRatio = sqrt(abs(rightSpeed / leftSpeed));
    float newRightSpeed;
    float newLeftSpeed;

    if (rightSpeed > 0){
      newRightSpeed = gCorrectionSpeed * rootSpeedRatio;
    }
    else if (rightSpeed < 0){
      newRightSpeed = -1 * gCorrectionSpeed * rootSpeedRatio;
    }
    if (leftSpeed > 0){
      newLeftSpeed = gCorrectionSpeed / rootSpeedRatio;
    }
    else if (leftSpeed < 0){
      newLeftSpeed = -1 * gCorrectionSpeed / rootSpeedRatio;
    }

    // If the distance is found to be different than the wanted by more than a
    // degree, call the driveWheels function with new speeds.
    if (encoderAverage() > degrees + 1.0){
      driveWheels(-1 * newRightSpeed, -1 * newLeftSpeed, encoderAverage() - degrees,
                  baseOffset / 10.0, cutoff / 10.0);
    }
    else if (encoderAverage() < degrees - 1.0){
      driveWheels(newRightSpeed, newLeftSpeed, degrees - encoderAverage(),
                  baseOffset / 10.0, cutoff/ 10.0);
    }
  }
  gCorrectionCounter = 0;
}


// Function to stop motors.
void stopMotor(int deceleration){
  sendByte(ACCELERATION, deceleration);
  sendByte(SPEEDR, 128);
  sendByte(SPEEDL, 128);
}


/////////////////////////////////
/////// Encoder Functions ///////
/////////////////////////////////


// Function to read and return the  value of an encoder as
// a long, takes the number of the encoder as an input.
long encoder(int encoderNumber){

  Wire.beginTransmission(MD25ADDRESS);

  if (encoderNumber == 1){
    Wire.write(ENCODER1);
  }

  if (encoderNumber == 2){
    Wire.write(ENCODER2);
  }

  Wire.endTransmission();
  Wire.requestFrom(MD25ADDRESS, 4);       // Request 4 bytes from MD25

  // Wait for 4 bytes to become available
  while(Wire.available() < 4) { /* do nothing */};

  long encoderValue = Wire.read();       // First byte for encoder 2, HH

  for (int i = 0; i < 3; i++){           //
    encoderValue <<= 8;                  // Read the next three bytes
    encoderValue += Wire.read();         //
  }

  return(encoderValue);                  //Return encoderValue
}


// Function that returns the absolute value of the average of the two encoders
float encoderAverage(){
  return( ( abs(encoder(1)) + abs(encoder(2)) ) / 2 );
}


// Function to set the encoder values to 0
void encodeReset(){
  sendByte(CMD,RESETENCODERS);
}


/////////////////////////////////
/////// Gripper Functions ///////
/////////////////////////////////


void closeGripper(int servoNumber) {

  if(is90secDone()) {
    launchRocket();
    return;
  }

  for (int pos = 5; pos <= gDegreesToOpen; pos += 1) {
    Servos[servoNumber].write(pos);
    delay(15);
  }

  delay(gDelayTime);

}

```

```

void openGripper(int servoNumber) {

  if(is90secDone()) {
      launchRocket();
      return;
  }

  for (int pos = gDegreesToOpen; pos >= 5; pos -= 1) {
    Servos[servoNumber].write(pos);
    delay(15);
  }

  delay(gDelayTime);

}


void turnGripperVertical(int servoNumber) {

  if(is90secDone()) {
      launchRocket();
      return;
  }

  //SERVO NO. 0
  if(servoNumber == 0) {
    for(int pos = 22; pos <= gDegreesToTurn; pos += 1) {
      ServosTurn[servoNumber].write(pos);
      delay(10);
    }
  }

  //SERVO NO. 1
  else if(servoNumber == 1) {
    ServosTurn[servoNumber].write(5);
  }

  //SERVO NO. 2
  else if(servoNumber == 2) {
    for(int pos = 5; pos <= 95; pos += 1) {
      ServosTurn[servoNumber].write(pos);
      delay(5);
    }
  }

  delay(gDelayTime);

}


void turnGripperHorizontal(int servoNumber) {

  if(is90secDone()) {
      launchRocket();
      return;
    }

  //SERVO NO. 0
  if(servoNumber == 0) {
    ServosTurn[servoNumber].write(22);
  }

  //SERVO NO. 1
  else if(servoNumber == 1) {
    ServosTurn[servoNumber].write(85);
  }

  //SERVO NO. 2
  else {
    ServosTurn[servoNumber].write(10);
  }

  delay(gDelayTime);

}


/////////////////////////////////
////// Obstacle Aboidance ///////
/////////////////////////////////


// Returns the distance away from either the front sensor when
// isForward = true, and the back sensor when isForward = false
float distance(bool isForward){

  float duration;
  float distance;

  if(isForward){
    digitalWrite(gFrontTrigPin, LOW);
    delayMicroseconds(2);
    // Sets the gTrigPin on HIGH state for 10 micro seconds
    digitalWrite(gFrontTrigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(gFrontTrigPin, LOW);
    // Reads the echoPin, returns the sound wave travel time in microseconds
    duration = pulseIn(gFrontEchoPin, HIGH);
    }

  else{
    digitalWrite(gBackTrigPin, LOW);
    delayMicroseconds(2);
    // Sets the gTrigPin on HIGH state for 10 micro seconds
    digitalWrite(gBackTrigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(gBackTrigPin, LOW);
    // Reads the echoPin, returns the sound wave travel time in microseconds
    duration = pulseIn(gBackEchoPin, HIGH);
    }

  // Calculating the distance
  distance = duration * 0.34 / 2;

  return(distance);

}
```

```


// Returns true if the robot is further away than distanceLimit from the
// either the front when isForward = true and the back when isForward = false
bool isClear(bool isForward, float distanceLimit){
  float currentDistance = distance(isForward);
  return(currentDistance > distanceLimit);
}


////////////////////////////////
/////// Rocket Functions ///////
////////////////////////////////


// Stops motor and returns true if 90 second have pased since the
// pull cord was pulled, returns false otherwise.
bool is90secDone(){
  if (millis() - gStartTime >= 90000){
    stopMotor(gDeceleration);
    delay(1000);
    return true;
  }
  else {
    return false;
  }
}


// Waits until 90 seconds have transcurred since the pull cord was
// launched, then launches the rocket by pulsing gRocketPin
void launchRocket(){

  while(is90secDone() == false){ /* Do nothing */}
  delay(1000);
  digitalWrite(gRocketPin, HIGH);
  delay(1000);
  digitalWrite(gRocketPin, LOW);
  exit(0);

}


////////////////////////////////
/////// Helper Functions ///////
////////////////////////////////


// Function that sends a value to a byte address
void sendByte( byte byteAddress, int value ){
  Wire.beginTransmission(MD25ADDRESS);
  Wire.write(byteAddress);                //'Write' to byteaddress
  Wire.write(value);                      //Send a value to that adress
  Wire.endTransmission();
}


// Function to convert a distance to a degree value
float distanceToDegrees(float distance){
  return(distance / gWheelDiameter / 3.1415 * 360);
}


// Function to find the degrees the wheels have to spin
// for a degree value of spinning on the spot
float onspotDegreesToWheelDegrees(float degrees){
  return(distanceToDegrees(degrees / 360 * 3.1415 * gWheelbase));
}


// No need for a loop function
void loop(){
}
```