Appendix A: Program Code

A.1 ROBOT.ino

```
// ROBOT.INO
// ALL METHODS AND PROGRAM CODE WRITTEN
// BY MATTHEW SMITH UNLESS OTHERWISE INDICATED
#undef main

/* THESE LIBRARIES ARE WRITTEN BY EXTERNAL AUTHORS
   BECAUSE OF THEIR SIZE LINKS TO THESE LIBRARIES
   ARE PROVIDED IN LIEU OF THEIR SOURCE CODE

   THE AFMotor LIBRARY IS AUTHORED BY ADAFRUIT INDUSTRIES[4]
   THE AFMotor LIBRARY ALLOWS THE ARDUINO TO COMMUNICATE WITH
   THE ADAFRUIT MOTOR CONTROLLER

   THE I2Cdev LIBRARY IS AUTHORED BY JEFF ROWBERG[5]
   THE I2Cdev LIBRARY SIMPLIFIES I2C COMMUNICATIONS
   THROUGH THE WIRE LIBRARY

   THE MPU6050 LIBRARY IS AUTHORED BY JEFF ROWBERG[6]
   THE MPU6050 LIBRARY ALLOWS THE OUTPUT OF THE MPU6050
   IMU TO BE EASILLY FETCHED AND PROCESSED

   THE Wire LIBRARY IS AUTHORED BY ARDUINO[7]
   THE Wire LIBRARY ALLOWS FOR I2C COMMUNICATIONS
   WITHOUT MANIPULATING BIT REGISTERS
*/
#include <AFMotor.h>
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
#include "Wire.h"
/* THIS INDICATES THE END OF EXTERNAL LIBRARIES
   ALL LIBRARIES FOLLOWING THIS ARE ORIGINAL CONTENT
*/

#include <PID.h>
#include <AdafruitWrapper.h>
#include <MotorController.h>

// comment out ADAFRUIT and uncomment L298N if using the L298N motor controller
#define ADAFRUIT
//#define L298N

#define MIN_SPEED 50 // minimum speed to avoid motor stiction
#define MAX_SPEED 255 // 255 corresponds to 100% duty

#if defined L298N
  #define ENA 9
  #define ENB 11
  #define IN1 8
  #define IN2 12
  #define IN3 10
```

```
  #define IN4 13
  MotorController motors(ENA, ENB, IN1, IN2, IN3, IN4, MIN_SPEED, MAX_SPEED);
#elif defined ADAFRUIT
  AdafruitWrapper motors(2, 3, MIN_SPEED, MAX_SPEED);
#endif

double setpoint = 176; // >180 is away from arduino cable
double PIDinput = 0.0, PIDoutput = 0.0;

uint8_t MPUinput[64];
uint8_t mpuPacketSize = 42;
volatile bool mpuHasInterrupt = false;

#define Kp 30
#define Ki 60
#define Kd 2

MPU6050 mpu;
PID pid(Kp, Ki, Kd, setpoint);

boolean haltExecution = false;

void MPUDataInterrupt() {
  mpuHasInterrupt = true;
}

double getPIDInput(uint8_t raw[]) {
  /* EVERYTHING BEYOND THIS COMMENT UNTIL INDICATED IS WRITTEN BY
     LUKA GABRIC[8]
     THIS CODE CONVERTS THE RAW DATA FROM THE IMU INTO A MEASURE OF
     THE ANGLE THE ROBOT MAKES WITH THE GROUND VIA A DATA MANAGEMENT
     PROCESSOR(DMP) ONBOARD THE IMU
  */
  Quaternion quart;
  VectorFloat gravityVec;
  float yawPitchRoll[3];

  mpu.dmpGetQuaternion(&quart, raw);
  mpu.dmpGetGravity(&gravityVec, &quart);
  mpu.dmpGetYawPitchRoll(yawPitchRoll, &quart, &gravityVec);
  return yawPitchRoll[1] * 180/M_PI + 180;
  /* THIS COMMENT INDICATES THE END OF PROGRAM CODE WRITTEN BY JEFF ROWBERG */
}

void handleInterrupt() {
  mpuHasInterrupt = false;

  if(mpu.getIntStatus() & 0x10) { // should never happen, clearing the
                                  // buffer takes long enough to crash the bot
    mpu.resetFIFO();
  } else {
    int fifoBufferSize = mpu.getFIFOCount();
    while (fifoBufferSize < mpuPacketSize) {
      fifoBufferSize = mpu.getFIFOCount(); // waits for the buffer to catch up
```

```
                                                // with the interrupt pin
    }
    while (fifoBufferSize > mpuPacketSize) { // reads freshest batch of data
      mpu.getFIFOBytes(MPUinput, mpuPacketSize);
      fifoBufferSize -= mpuPacketSize;
    }
  }
  PIDinput = getPIDInput(MPUinput); // converts the raw MPU data to an angle
  PIDoutput = pid.returnOutput(PIDinput);
  motors.set(-PIDoutput);
}

/*
This function runs while the user holds the robot vertically upright
for a few seconds so that the robot knows what setpoint to balance about.
This function runs until the setpoint is printed to serial.

This function determines the default angle about which the robot should balance
by averaging the IMU values over a few seconds where the robot is held level by the use
*/
void calibrate(double & setpoint) { //AUTHOR: ARJUN BALI
  double sum;
  for (int i = 0; i < 1000; i++)
  {
    while(!mpuHasInterrupt);

    mpuHasInterrupt = false;

    while (mpu.getFIFOCount() < mpuPacketSize);
    mpu.getFIFOBytes(MPUinput, mpuPacketSize);
    PIDinput = getPIDInput(MPUinput);
    sum += PIDinput;
  }
  setpoint = sum/1000;
  Serial.println(setpoint);
}

/*
  Capital letters increment
  Lowercase letters decrement
  S - setpoint
  P - proportional term
  I - integral term
  D - derivative term

  Q - halts execution
  */
void handleSerial(char in) {
  switch(in) {
    case 'S':
    setpoint += 0.1;
    break;
  case 's':
    setpoint -= 0.1;
```

```
    break;
  case 'P':
    pid.incP(1);
    break;
  case 'p':
    pid.incP(-1);
    break;
  case 'I':
    pid.incI(0.5);
    break;
  case 'i':
    pid.incI(-0.5);
    break;
  case 'D':
    pid.incD(0.2);
    break;
  case 'd':
    pid.incD(-0.2);
    break;
  case 'Q':
    haltExecution = true;
  }
}

void setup() {
  /* EVERYTHING BEYOND THIS COMMENT UNTIL INDICATED IS WRITTEN BY
     JEFF ROWBERG[9]
  */
    // join I2C bus (I2Cdev library doesn't do this automatically)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
      Wire.begin();
      TWBR = 24; // 400kHz I2C clock
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
    #endif

    // initialize serial communication
    // (115200 chosen because it is required for Teapot Demo output, but it's
    // really up to you depending on your project)
    Serial.begin(19200);
    while (!Serial); // wait for Leonardo enumeration

    // initialize device
    Serial.println(F("Initializing I2C devices..."));
    mpu.initialize();

    // verify connection
    Serial.println(F("Testing device connections..."));
    Serial.println(mpu.testConnection() ? F("MPU6050 connection successful")
                 : F("MPU6050 connection failed"));

    // load and configure the DMP
    Serial.println(F("Initializing DMP..."));
    uint8_t devStatus = mpu.dmpInitialize();
```

```
    mpu.setXGyroOffset(0);
    mpu.setYGyroOffset(0);
    mpu.setZGyroOffset(0);
    mpu.setZAccelOffset(0);

    // make sure it worked (returns 0 if so)
    if (devStatus == 0)
    {
        // turn on the DMP, now that it's ready
        Serial.println(F("Enabling DMP..."));
        mpu.setDMPEnabled(true);

        // enable Arduino interrupt detection
        Serial.println(F("Enabling interrupt detection..."));
        attachInterrupt(0, MPUDataInterrupt, RISING);
        mpuHasInterrupt = mpu.getIntStatus();

        // set our DMP Ready flag so the main loop() function knows it's ready
        Serial.println(F("DMP ready!"));
        mpuHasInterrupt = true;

        // get expected DMP packet size for later comparison
        mpuPacketSize = mpu.dmpGetFIFOPacketSize();
        Serial.println(mpuPacketSize);
    }
    else
    {
        // ERROR!
        // 1 = initial memory load failed
        // 2 = DMP configuration updates failed
        // (if it's going to break, usually the code will be 1)
        Serial.print(F("DMP Initialization failed (code "));
        Serial.print(devStatus);
        Serial.println(F(")"));
    }

    /* THIS COMMENT INDICATES THE END OF PROGRAM CODE WRITTEN BY JEFF ROWBERG */

    pid.setMaxSpeed(MAX_SPEED);
    calibrate(setpoint);

    Serial.println("setup() complete!");
}


void loop() {
  while (!mpuHasInterrupt) { // wait for fresh data
    if(abs(PIDinput-180) > 45) { // robot has fallen over||been ordered to stop
      haltExecution = true;
    }
    if(Serial.available())
      handleSerial((char)Serial.read());
  }
```

```
  handleInterrupt();
}

int main() {
  init(); //default arduino setup method
  setup();

  while(!haltExecution)
    loop();
  Serial.println("Main loop terminated");
  motors.set(0);
  #ifdef ADAFRUIT
    motors.cleanUp();
  #endif
  return 0;
}
```

A.2 PID.cpp

```
// PID.CPP
#include "Arduino.h"
#include <PID.h>

PID::PID(double Kp, double Ki, double Kd, double & setpoint) {
    _maxSpeed = 100;
    _integralTerm = 0.0;
    _samplingPeriod = 10;

    _Kp = Kp;
    _Ki = Ki;
    _Kd = Kd;

        _lastUpdate = millis()-_samplingPeriod;
    _setpoint = setpoint;
    _freshData = false;
}

double PID::returnOutput(double input) {
        double timeDiff = (millis() - _lastUpdate);

        if(timeDiff >= _samplingPeriod) { // samplingPeriod = the period of the IMU
                _lastUpdate = millis();
                _freshData = true; // let the motor controller know the PID has data
                _lastError = input;

                timeDiff /= 1000; //convert ms into seconds

                double error = _setpoint - input;
                _integralTerm += error*timeDiff;
                double derivative = (input - _lastError)/timeDiff;
                double output = _Kp * error + _Ki * _integralTerm + _Kd * derivative;

                if(abs(output) > _maxSpeed)
                        output = output * _maxSpeed / abs(output);
                return output;
        } else {
                _freshData = false;
                return 0.0;
        }
}

void PID::setMaxSpeed(double max) {
        _maxSpeed = max;
}
void PID::incP(double inc) {
        _Kp += inc;
}
void PID::incI(double inc) {
        _Ki += inc;
}
void PID::incD(double inc) {
```

```
        _Kd += inc;
}

bool PID::freshData() {
        return _freshData;
}

void PID::setSetpoint(double setpoint) {
        _setpoint = setpoint;
}
```

A.3 PID.h

```
// PID.H
#ifndef PID_h
#define PID_h

class PID
{
  private:
    double _setpoint;
        int _samplingPeriod;
        double _Kp;
    double _Ki;
    double _Kd;
        bool _freshData;

        unsigned long _lastUpdate;
        double _integralTerm, _lastError;
        double _maxSpeed;

  public:
    PID(double Kp, double Ki, double Kd, double & setpoint);
        double returnOutput(double input);
    void setMaxSpeed(double max);
    void setSetpoint(double setpoint);
    void incP(double inc);
    void incI(double inc);
    void incD(double inc);
    bool freshData();
};
#endif
```

A.4 AdafruitWrapper.cpp

```
// ADAFRUITWRAPPER.CPP
#include "Arduino.h"
#include "AFMotor.h"
#include "AdafruitWrapper.h"

AdafruitWrapper::AdafruitWrapper(int leftMotorPort, int rightMotorPort,
        int minSpeed, int maxSpeed) {
        int leftMotorFreq, rightMotorFreq;
        leftMotorFreq = rightMotorFreq = MOTOR12_1KHZ;

        if(leftMotorPort==3 || leftMotorPort==4)
                leftMotorFreq = MOTOR34_1KHZ;
        if(rightMotorPort==3 || rightMotorPort==4)
                rightMotorFreq = MOTOR34_1KHZ;


        _leftMotor = new AF_DCMotor(leftMotorPort, leftMotorFreq);
        _rightMotor = new AF_DCMotor(rightMotorPort, rightMotorFreq);

        _minSpeed = minSpeed;
        _maxSpeed = maxSpeed;
}

/*
// This function communicates with the Adafruit Motor Controller's
proprietary motor controller library to adjust the duty cycle of the signal
powering the motors. This function also manipulates the provided speed by
scaling and offsetting it to help overcome motor stiction.
(Written by Justin Lim)
*/
void AdafruitWrapper::set(int speed) {
        speed *= (_maxSpeed - _minSpeed);
        speed /= _maxSpeed;

        if(speed > 0) {
                speed += _minSpeed;
        } else if(speed < 0) {
                speed -= _minSpeed;
        }

        _leftMotor->setSpeed(abs(speed));
        _rightMotor->setSpeed(abs(speed));
        if(speed > 0) {
        _leftMotor->run(FORWARD);
        _rightMotor->run(FORWARD);
        } else {
        _leftMotor->run(BACKWARD);
                _rightMotor->run(BACKWARD);
        }
}

void AdafruitWrapper::cleanUp() {
```

```
        delete _leftMotor;
        delete _rightMotor;
}
```

A.5 AdafruitWrapper.h

```
// ADAFRUITWRAPPER.H
#ifndef AdafruitWrapper_h
#define AdafruitWrapper_h

#include "AFMotor.h"

class AdafruitWrapper {
        private:
                AF_DCMotor *_leftMotor, *_rightMotor;
                int _minSpeed, _maxSpeed;
        public:
                AdafruitWrapper(int leftMotorPort, int rightMotorPort,
                                                int minSpeed, int maxSpeed);
                void set(int speed);
                void cleanUp();
};

#endif
```

A.6 MotorController.cpp

```
// MOTORCONTROLLER.CPP
#include "MotorController.h"
#include "Arduino.h"

MotorController::MotorController(int pinA, int pinB, int pinFwdA, int pinRevA,
         int pinFwdB, int pinRevB, int minSpeed, int maxSpeed) {
        _pinA = pinA;
        _pinB = pinB;

        _pinFwdA = pinFwdA;
        _pinRevA = pinRevA;
        _pinFwdB = pinFwdB;
        _pinRevB = pinRevB;

        _minSpeed = minSpeed;
        _maxSpeed = maxSpeed;

        pinMode(_pinA, OUTPUT);
        pinMode(_pinB, OUTPUT);
        pinMode(_pinFwdA, OUTPUT);
        pinMode(_pinRevA, OUTPUT);
        pinMode(_pinFwdB, OUTPUT);
        pinMode(_pinRevB, OUTPUT);

        _last = 0;
        _delay = 0;
}

void MotorController::set(int speed) {
        if(speed > 0) {
                fwdA();
                fwdB();
        } else {
                revA();
                revB();
        }

        speed *= (_maxSpeed - _minSpeed);
        speed /= _maxSpeed;

        if(speed > 0) {
                speed += _minSpeed;
        } else if(speed < 0) {
                speed -= _minSpeed;
        }

        analogWrite(_pinA, speed);
        analogWrite(_pinB, speed);
}

void MotorController::fwdA() {
        digitalWrite(_pinFwdA, HIGH);
```

```
        digitalWrite(_pinRevA, LOW);
}

void MotorController::revA() {
        digitalWrite(_pinFwdA, LOW);
        digitalWrite(_pinRevA, HIGH);
}

void MotorController::fwdB() {
        digitalWrite(_pinFwdB, HIGH);
        digitalWrite(_pinRevB, LOW);
}

void MotorController::revB() {
        digitalWrite(_pinFwdB, LOW);
        digitalWrite(_pinRevB, HIGH);
}
```

A.7 MotorController.h

```
// MOTORCONTROLLER.H
#ifndef MotorController_h
#define MotorController_h

#include "Arduino.h"

class MotorController {
        private:
                int _pinA, _pinB, _pinFwdA, _pinRevA, _pinFwdB, _pinRevB, _last,
                        _delay, _minSpeed, _maxSpeed;
        public:
                MotorController(int pinA, int pinB, int pinFwdA, int pinRevA,
                int pinFwdB, int pinRevB, int minSpeed, int maxSpeed);
                void set(int speed);
                void fwdA();
                void revA();
                void fwdB();
                void revB();
};

#endif
```