Individual Lab Report - 1

Autonomous Reaming for Total Hip Replacement

HIPSTER | ARTHuR

Sundaram Seivur

Team C:

Kaushik Balasundar | Parker Hill | Anthony Kyu

Sundaram Seivur | Gunjan Sethi

February 10th 2022

Carnegie Mellon University

The Robotics Institute

Contents

1 Individual Progress

1.1 Sensor and Motors Lab

My responsibilities in the Sensors and Motors Lab was to implement velocity control of the DC motor using PID control. This meant that I took readings from the encoder to count the number of ticks for each revolution of the main shaft and converted that to RPM. Given an input signal from a sensor or the GUI, the motor converges to the given input RPM by reducing error. I was also responsible for interfacing the readings from the Ambient Light Sensor, display those readings as Lux output and to run all the motors based on the output reading. In the case of controlling the velocity of the DC motor with the Ambient Light Sensor, it meant that with increasing value of light (lux), the speed of rotation of the motor increased and vice versa.
Additionally, I was also entrusted with integrating all the hardware together which included connecting all the motors and sensors together in a tidy fashion.

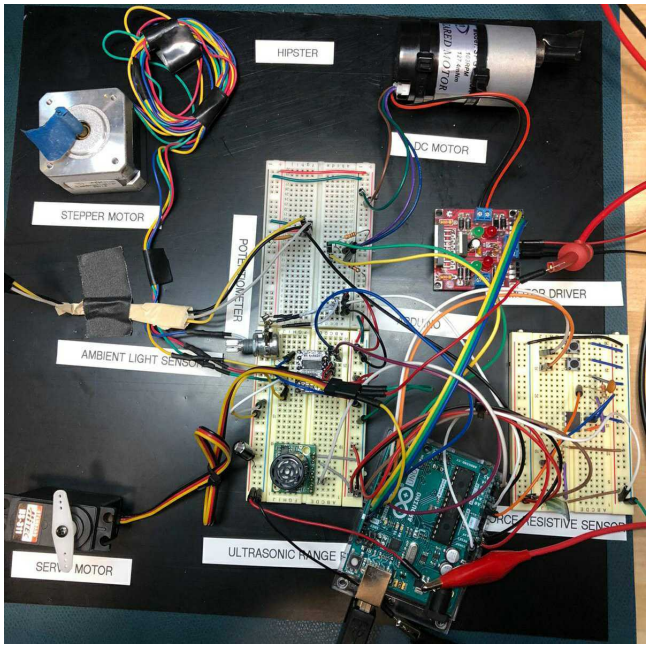

Figure 1: Integrated circuited with motors and sensors

1.1.1 Ambient Light Sensor

The Ambient Light Sensor by modern devices is an analog sensor that outputs a value between 0 and 1023 with a Sharp GA1A1S201WP surface-mount ambient light sensor. The range of the sensor is between 3 to 55000 lux and I found the maximum value (in the lighting conditions in B506) to be close to 930. The sensor is prone to external noise and the output fluctuates rapidly without any filter. Hence, I decided to implement a moving average filter and use a 1nF capacitor to account for voltage fluctuations. A moving average filter could lead to lag in the output signal, but I was able to balance the lag and reduce lag effectively by tuning the window size. The capacitor helped in reducing noise partially, but the performance was significantly improved after applying the moving average filter. Once I got stable output, I needed to convert the analog output to

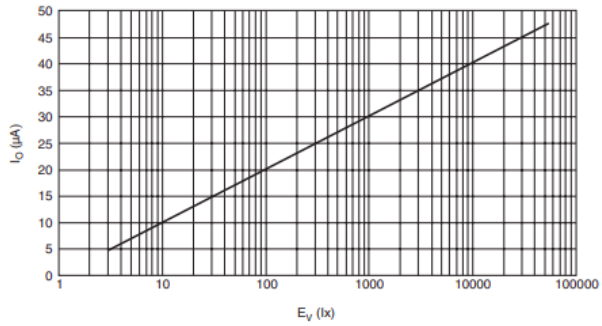

Figure 2: Current vs Illuminance of ambient light sensor

physical units, which in this case would be Lux. In this regard, I referred to the datasheet of the sensor to find a plot that relates current (mA) to illuminance (lux) as seen in Fig. **??** The plot shows a logarithmic relationship between current and illuminance throughout the operating range of the sensor. To convert an analog output to Lux, I mapped the output to voltage output by mapping 0V-5V to 0-1023. This voltage was then converted to a current signal by diving the voltage value by a 10k resistor. This current output was then mapped to Lux using the graph in the datasheet. To validate the datasheet plot, I also took samples of the Lux output of the sensor and compared it with the lux meter output of more sophisticated sensors in our smartphones. The output was in close approximation with the output from the smartphone with an error of $\pm$ 10 lux.

1.1.2 DC Motor : Velocity Control

Velocity control for the DC motor was implemented using PID control. I wrote a function to count the number of ticks captured by both the hall effect sensors present in the encoder. This counting was done by attaching the Channel A and Channel B of the encoder to interrupt pins in Arduino. I converted the number of ticks counted per second to number of motor shaft revolutions per second by dividing the encoder value by 90 (30:1 gear ratio with 3 pulses per second).
To tune the RPM of the motor shaft, I wrote a function to calculate the control signal by computing the error between current velocity and desired velocity. This control signal was then sent as an PWM signal to the motor for velocity compensation.

1.2 MRSD Project

My primary contribution to the ARTHUR project (Autonomous Reaming for Total Hip Replacement) has been with writing and integrating controller files for the robot arm to be controlled in simulation. I have simultaneously also worked on developing a node to publish joint trajectory commands for the robot joints to track and reach a goal state. I am currently working on publishing a trajectory to the arm in Gazebo from Moveit which takes a start and end point as input. In the start of the semester I also spent little time on writing some parts of the URDF file as this file given to us by our sponsors was incomplete.

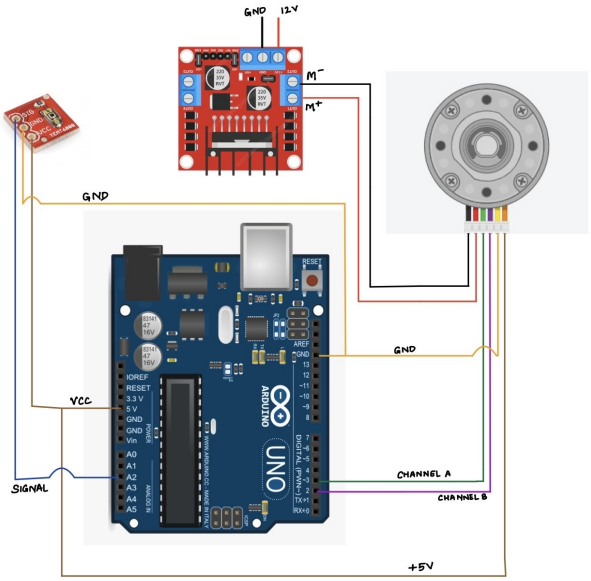

Figure 3: DC motor and Ambient Light Sensor

2 Challenges

2.1 Sensors and Motors Lab

The main challenges we faced was while integrating the hardware with the software and to get all the motors and sensors working at once. We implemented two push buttons, one for changing motors and the other for changing sensors. This lead to us implementing a state machine with multiple switch cases. The switch cases had a lot of edge cases which we did not foresee and had to debug progressively. For example, when we switched from stepper motor to servo motor, the DC motor started rotating as it might have gotten a stray signal.
Secondly, we faced issues with getting velocity control of the DC motor working with all the hardware integrated. All the motors were attached to all the available PWM pins on arduino which lead to a large amouunt of current being drawn at all times. As the stepper motor also drew high currents, this led to voltage fluctuations in the DC motor. These voltage fluctuations caused the DC motor to have speed fluctuations with jittery motion. We accounted for this by better tuning the PID parameters, reducing code latency and diving power to the circuit from dual sources.

2.2 MRSD Project Challenges

The most important issues we faced in the capstone project was with receiving hardware from our sponsors. Placing an order for the arm and its eventual dispatch was delayed. We received the arm two weeks post our initial estimate. We had taken this risk into account and had planned a mitigation strategy by spending the first few weeks working on simulation.
An unforeseen challenge we are currently facing is inadequate support from Kinova to integrate any external code with the arm. Since the arm is a custom model which hasn't been released commercially yet, most APIs are still under development. The arm can't be interfaced with ROS, Python or CPP. We have discussed this problem with our sponsors and they have requested for a

Kinova Gen-3 arm for us to use till the other arm becomes usable.

3 Team Work

3.1 Anthony Kyu

3.1.1 Sensors and Motors Lab

Anthony set up the state machine for the arduino code, worked on the Force Sensitive Resistor and Servo motor. He also setup the code to be integrated with the GUI.

3.1.2 MRSD Project

He designed the camera mount, contributed in fabricating the mount and has been spending time in researching control algorithms for the project.

3.2 Kaushik Balasundar

3.2.1 Sensors and Motors Lab

Kaushik worked with the potentiometer and postion control of the DC motor. He also helped with the system integration

3.2.2 MRSD Project

He initially spent most of his time in setting up a docker environment and code repository for the project. He then worked on cleaning the URDF file and setup the simulation environment. He is currently working on integrating Open3D with ROS for registration.

3.3 Gunjan Sethi

3.3.1 Sensors and Motors Lab

Gunjan developed the GUI for the lab and also helped in integrating the sensors and motors with the GUI. She spent some time debugging the program as well.

3.3.2 MRSD Project

As part of the perception team, she has been setting up the Atracys camera, linked the camera with ROS and wrote a ROS node to discover the camera and markers. She is also managing Jira for our project.

3.4 Parker Hill

3.4.1 Sensors and Motors Lab

Parker worked on the stepper motor and ultrasonic range finder sensor as part of the lab. He also contributed in the hardware and system integration.

3.4.2 MRSD Project

He contributed in fabricating the camera mount, designing connectors for the markers and is learning ROS.

4 Future Plan

The sensors and motors lab taught me the importance and difficulties of system integration. When multiple subsystems are involved in a project, the performance of some or all of the individual systems can get compromised if not integrated well. The bugs in our code also made me realize the importance of unit testing each function thoroughly for all edge cases before integrating them as a whole. Although the actuators and sensors do not have direct relevance to my capstone project, I have systems integration lessons to be carried forward to my capstone project.

Our immediate plan as a team is to find leads to get a manipulator temporarily that can suit our needs in terms of force and position accuracy attainable by the arm. In the meantime, we would continue to work on our individual tasks as planned in the schedule. We will also look at modifying the schedule to take into account the delays caused by hardware and software incompatibility of the arm provided to us by our sponsors. As the Motion planning lead, I will develop scripts for interfacing MoveIt with the arm, such that the trajectories generated by MoveIt are published to our system with low latency.
I am personally taking responsibility to keep the team on track with the schedule. I will continue to contribute in helping the multiple subsystems to integrate with each other seamlessly.

5 Quiz

5.1 ADXL335

- The sensor has a typical range of $\pm 3.6g$.
- The dynamic range of the sensor is 3.6g - (-3.6g) = 7.2g.
- C_{dc} is a 0.1μ F capacitor that is attached close to the input supply pin to decouple the accelometer from noise on the power supply.
  When several devices share a power supply via a common path, an increase or decrease in the current drawn by one of the devices can lead to voltage changes and transient currents large enough to affect the operation of other devices. A decoupling capacitor, like C_{dc} in ADXL335 provides a path for transients currents to pass through instead of the common impedance of the circuit. The capacitor acts as a local energy storage device.
- $V_{out} = 1.5V + (0.3V/g) \times a$
- The largest expected nonlinearity = 0.3% of full scale
  Typical full scale = $\pm$ 3.6g
  Therefore, the largest expected nonlinearity = 0.003 $\times$ 7.2 = $\pm$ 0.0216g
- The bandwidth of the X and Y axes are 0.5Hz to 1600Hz.

- The noise density of the ADXL335 is given as $150\mu g$
  RMS Noise = Noise density * $\sqrt{BW * 1.6}$
  RMS Noise = $150 \times \sqrt{25 * 1.6} = 948.68\mu g$
- The ADXL335 measures acceleration (g) and is expected to have an $0g$ when kept on a perfectly flat surface, with an supply voltage of 3V. We can take repeated measurements along the flat surface to measure acceleration. We then calculate the mean of the recorded measurements, subtract the mean from the recorded acceleration values and then square all the resulting values. Finally, we find a mean of all the squared values by adding them and dividing by the total number of measurements to get the RMS error value

5.2 Signal Conditioning

- The moving average filter requires maintaining a history of data based on the window size. This leads to a lag in the measured data
- The filter can overlook certain important characteristics of the curve my averaging it out. If a smaller window is chosen for this reason, then it would allow noise to pass in

Name at least two problems you might have in using a median filter.

- The computational cost of a median filter is very high
- They are prone to get affected by small signal-to-noise ratios

5.3 Opamps

In the following questions, you want to calibrate a linear sensor using the circuit in Fig. 1 so that its output range is 0 to 5V. Identify in each case: 1) which of V1 and V2 will be the input voltage and which the reference voltage; 2) the values of the ratio Rf/Ri and the reference voltage. If the calibration can't be done with this circuit, explain why. Your uncalibrated sensor has a range of -1.5 to 1.0V (-1.5V should give a 0V output and 1.0V should give a 5V output). Your uncalibrated sensor has a range of -2.5 to 2.5V (-2.5V should give a 0V output and 2.5V should give a 5V output).

5.3.1 Case 1:

V_1 is the reference voltage and V_2 is the input voltage since the amplifier is non-inverting

$$\frac{V_1 - V_2}{R_i} = \frac{V_2 - V_o ut}{R_f} \implies V_{out} = (V_2 - V_1)\frac{R_f}{R_i} + V_2 \tag{1}$$

The above equation has two unknowns $\frac{R_f}{R_i}$ and V_{ref}. Substituting $(V_{out} = 0V, V_2 = -1.5V)$ and $(V_{out} = 5V, V_2 = 1V)$ we get,

$$-1.5 = (V_{ref} + 1.5)\frac{R_f}{R_i} \tag{2}$$

$$4 = (1 - V_{ref})\frac{R_f}{R_i} \tag{3}$$

Solving these equations we get,

$$\frac{R_f}{R_i} = 1,\ V_1 = V_{ref} = -3V \tag{4}$$

Therefore, the calibration of this sensor is achievable by this OpAmp.

5.3.2 Case 2:

$$V_{out} = (V_2 - V_1)\frac{R_f}{R_i} + V_2 \tag{5}$$

Substituting $(V_{out} = 0V, V_2 = -2.5V)$ and $(V_{out} = 5V, V_2 = 2.5V)$ we get,

$$\frac{R_f}{R_i} = 0 \tag{6}$$

This is not viable as R_f cannot be zero to maintain the functionality of the OpAmp. Hence, calibration of this sensor is not achievable by this OpAmp.

5.4 Control

- The equation to control a Dc motor with PID is:

$$u = K_p * error + K_d * error + \int K_i * error\ dt \tag{7}$$

  where Kp stands for proportional term, Kd stands for the derivative term and Ki stands for the intergral. The enconder is used to control the position of the main shaft of the DC motor. The encoder counts the number of the number of ticks or in other words the encoder outputs a sequence of pulses, which gets mapped to the DC motor position. Given a desired position, the difference or the error between the desired and current position is fed to the PID equation and a control signal is computed as the output.
  In our system, we computed the error my counting the number of ticks the encoder has intercepted from the zero position. Since the encoder has a resolution of 90 ticks per main shaft revolution, that means each ticks corresponds to a rotation of 4 degrees. Given the error between the current position and desired position in degrees, the difference is fed to the PID function. The control signal is mapped to values between 0 and 255 and passed to the DC motor. This allows the motor to move to a user commanded position.

- If the system is sluggish, I would use the P term as this term accounts for the overshoot of the current signal from the desired signal. If the system is deemed sluggish, that means that the current signal doesn't reach the desired output. The P term will allow the signal to rise till the desired output and possibly even overshoot. The P term behaves as the spring in a mass spring damper system.

- After applying the P term, if the system still has steady-state error then I will use the integral, also called the I term in PID. The I term acts as a memory for the system by integrating the

error of the system over time and forcing the system to close down the steady-state error.

- After applying the P and I terms, if the system still has overshoots then I would be applying the D term that acts as a damper for the system. The D term predicts the value of the control signal by computing the slope of the error

6 Code

```
    /** Libraries */



/** Macros */
#define LIGHTSENSORPIN A2 //Ambient light sensor reading
#define light_window 50

/** Pin Assignments */

//Motor encoderPins
const int ENCA = 2;
const int ENCB = 3;
//Motor signal pins
const int PWM = 9;
const int IN1 = 11;
const int IN2 = 12;


/** Global Variables */

// Ambient Light Sensor Variables
int a = 0;
int a_analog = 0;
float sum = 0;
float sum_analog = 0;
float readings[light_window];
float readings_analog[light_window];
float avg = 0;
float avg_analog = 0;
float lux;
int lux_analog;
long lightTimer = 0;

int m = 0;
float sum_motor = 0;
```

```
float readings_motor[light_window];
float avg_motor = 0;

// DC Motor Velocity Control Variables
int posPrev = 0;
float previousError = 0;
char incomingByte;
volatile int encoderValue = 0;
float edot = 0;
float proportional = 0.5; //2; //k_p = 0.5
float integral = 0.55; //3 //k_i = 3
float derivative = 0.0001;//0.12; //k_d = 1
float controlSignal = 0; //u - Also called as process variable (PV)
float errorIntegral = 0;
float deltaT;
//int pos;
long previousTime;
long currentTime;


// Setup*************************************
void setup()
{
  // Attach pins
  pinMode(FSR_Pin, INPUT);
  pinMode(sensorPushButton, INPUT_PULLUP);
  pinMode(motorPushButton, INPUT_PULLUP);
  attachPCINT(digitalPinToPCINT(sensorPushButton), changeSensor, FALLING);
  attachPCINT(digitalPinToPCINT(motorPushButton), changeMotor, FALLING);
  pinMode(USRF_Pin, INPUT);
  pinMode(LIGHTSENSORPIN,  INPUT);


  // Motor setup
  motorInput = 0;
  setupServo(myservo, servoPin);
  stepperTimer = micros();
  stepper.begin(Stepper_RPM, Stepper_Microsteps);
  pinMode(ENCA, INPUT);
  pinMode(ENCB, INPUT);
  pinMode(PWM, OUTPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  attachInterrupt(digitalPinToInterrupt(ENCA), encoder, RISING);
  currentTime = micros();
```

```
  previousTime = micros();
}


// loop**************************************************
void loop()
{

  if (Serial.available()) {
  //Read ambient light sensor - Lux output for GUI
  if ((micros() - lightTimer) > 100000) {
    lux = ambientLightRead();
    lux_analog = ambientLightAnalog();
    lightTimer = micros();
  }

  // Drive chosen motor
  switch (motorState)
  {
    case SERVO:

    case STEPPER:

    case DC_MOTOR_POS:

    case DC_MOTOR_VEL:
      if (sensorState != GUI)
      {
        motorInput = map(motorInput, 0, 1023, 0, 100);
        //        Serial.println(motorInput);
      }

      int pos_vel = encoderValue;

      long currentTime = micros();
      deltaT = ((float) (currentTime - previousTime) / 1000000);

      if (deltaT > 0.01)
      {
        float velocity = (pos_vel - posPrev) / deltaT;
        posPrev = pos_vel;
        previousTime = currentTime;

        float v1 = velocity * 60 / 90;
        float error = motorInput - v1;
```

```
        float u = calculatePIDVel(error);

        int dir = 1;
        if (u < 0)
        {
          dir = -1;
        }
        int pwr = (int) fabs(u);
        if (pwr > 255)
        {
          pwr = 255;
        }
        else if (pwr < 0)
        {
          pwr = 0;
        }
        setMotor(dir, pwr, PWM, IN1, IN2);

        sum_motor = sum_motor - readings_motor[m];
        readings_motor[m] = v1;
        sum_motor =  sum_motor + v1;
        m = (m + 1) % light_window;
        avg_motor = sum_motor / light_window;
      }
      break;
  }



}

// Sensor Functions*****************************

//Ambient Light sensor function
float ambientLightRead() {
  sum = sum - readings[a];
  float reading = analogRead(LIGHTSENSORPIN);
  readings[a] = reading;
  sum =  sum + reading;
  a = (a + 1) % light_window;
  avg = sum / light_window;

  float volt = (avg * 5.0) / 1023.0;
  //      Serial.print(" Volt:"); Serial.print(volt);
```

```
  float curr = pow(10, 6) * (volt / 100000);
  //      Serial.print(" Curr:"); Serial.print(curr);

  float lux = pow(10, 0.1 * curr);
  //      Serial.print(" Lux: "); Serial.println(lux);
  return lux;
}

//Ambient sensor analog output
float ambientLightAnalog() {
  sum_analog = sum_analog - readings_analog[a_analog];
  float reading_analog = analogRead(LIGHTSENSORPIN);
  readings_analog[a_analog] = reading_analog;
  sum_analog =  sum_analog + reading_analog;
  a_analog = (a_analog + 1) % light_window;
  avg_analog = sum_analog / light_window;
  //    Serial.print("Avg. Reading: "); Serial.print(avg_analog);
  return avg_analog;
}



// Motor Functions*******************************


//DC Motor Velocity Control

//Velcoity Control main PID Loop

void velocityControl() {

  int pos = encoderValue;
  float velocity = (pos - posPrev) / deltaT;
  posPrev = pos;
  previousTime = currentTime;

  float v1 = velocity * 60 / 90;

  float error = motorInput - v1;
  float u = calculatePIDVel(error);

  int dir = 1;
  if (u < 0) {
    dir = -1;
  }
```

```
  int pwr = (int) fabs(u);
  if (pwr > 255) {
    pwr = 255;
  }
  else if (pwr < 0) {
    pwr = 0;
  }
  setMotor(dir, pwr, PWM, IN1, IN2);

}

//Velcity control PID Function

float calculatePIDVel(float errorValue)
{
  edot = (errorValue - previousError) / deltaT; //edot = de/dt - derivative term
  errorIntegral = errorIntegral + (errorValue * deltaT);
  Serial.println(errorIntegral);
  if (errorIntegral > 480) {
    errorIntegral = 480;
  } else if (errorIntegral < -480) {
    errorIntegral = -480;
  }
  controlSignal = (proportional * errorValue) + (derivative * edot)
  + (integral * errorIntegral); //final sum, proportional term also calculated
  if (controlSignal > 260) {
    controlSignal = 260;
  } else if (controlSignal < -260) {
    controlSignal = -260;
  }
  previousError = errorValue; //save the error for the next
  iteration to get the difference (for edot)
  Serial.println(controlSignal);
  return controlSignal;
}

//Velocity and Position Control Encoder

void encoder() {
  if (digitalRead(ENCB) == HIGH) // if ENCODER_B is high increase the count
    encoderValue++; // increment the count

  else // else decrease the count
    encoderValue--;  // decrement the count
```

```
}

//General Motor Function

void setMotor(int dir, int pwmVal, int pwm, int in1, int in2)
{
  //  Serial.print(pwmVal);
  analogWrite(pwm, pwmVal); // Motor speed
  if (dir == 1) {
    // Turn one wayb
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
  }
  else if (dir == -1) {
    // Turn the other way
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
  }
  else {
    // Or dont turn
    digitalWrite(in1, LOW);
    digitalWrite(in2, LOW);
  }
}
```