Ian Hartwig

Team B: No Name

Teammates: Ian Rosado, Stephanie Chen, Trevor Decker

ILR 02

Feb. 12, 2015

Individual Progress

I contributed to the motor lab by writing the majority of the firmware on the arduino. I implemented control of the servo, stepper, and dc motor using the sensor input we developed in the sensors lab. The completed hardware system can be seen in figure 1. I primarily used the Arduino libraries out of convenience, including the servo library, pid library, and the PJRC encoder library. Our firmware implements 4 modes of operation defining which motors are controlled by which inputs. The modes can be found in table 1.

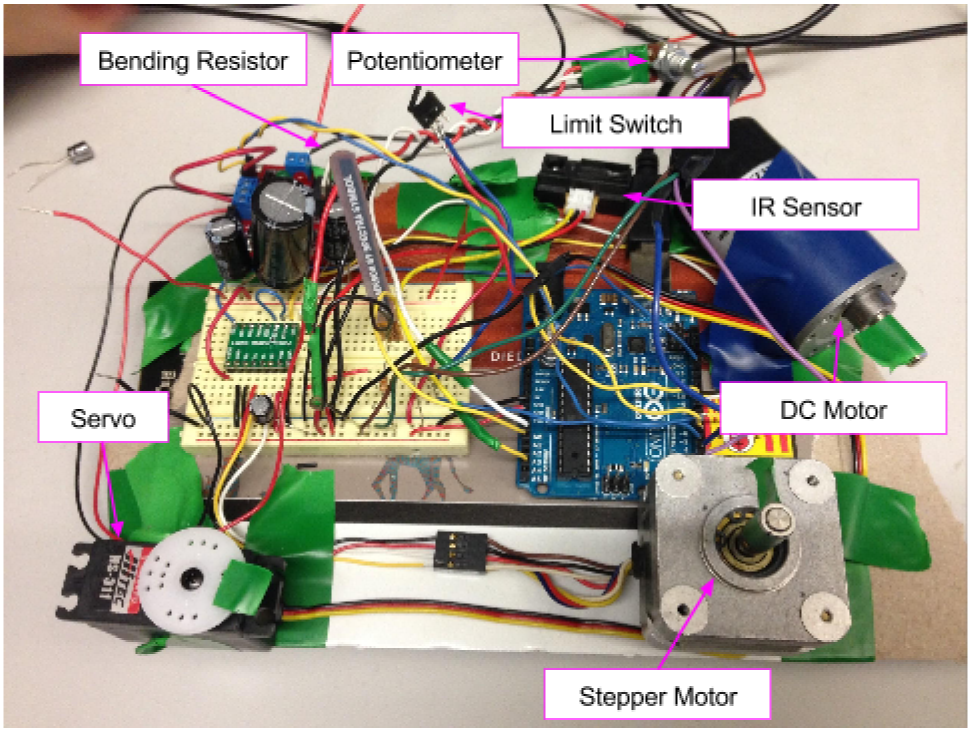

Figure 1: Motor Lab System at Demo Time.

Control of our servo is implemented using the Arduino servo library. The servo library uses the ATmega's 16 bit timer 1 to generate a 20ms period PWM signal. We filter the bend sensor input and directly write the value out to the servo. The stepper is controlled by assuming 0 position (in ticks) at startup and moving the motor (+) or (-) the desired number of ticks by pulsing into the provided A4988 stepper controllers. The setpoint is either the moving-average smoothed IR sensor value (or a computer interface value). DC motor control combines the readings from the encoder (hall effects) on the back of the motor with the L298N driver board provided. Position and velocity control use the Arduino PID library to convert the feedback and setpoint to a PWM output to the motor driver inputs. See the control firmware in Appendix A.

Table 1: Firmware Modes

	Servo	Stepper	DC Motor
Sensor Input, Motor Position	bend sensor -> position	IR sensor -> position	potentiometer -> motor position
Sensor Input, Motor Velocity	bend sensor -> position	IR sensor -> position	potentiometer -> motor velocity
Computer Input, Motor Position	computer input -> position	computer input -> position	computer input -> position
Computer Input, Motor Velocity	computer input -> position	computer input -> position	computer input -> velocity

Teamwork

Ian Rosado and Stephanie Chen did much of the electronics assembly for this lab. Ian populated the motor controller board, and Stephanie wired up the breadboard and sensors. They have also been working on the mechanical designed for the telescopic arm and gripper, respectively. For this lab, Trevor primarily took on the MATLAB gui programming, as he is proficient in that area. He built a gui that supports live data streaming in any firmware mode and allows computer control of the sensors when the system is in the right mode as well as a serial protocol to send and read control values.

Challenges

We are still investigating the best way to configure motors, and power transfer in our design. We need to lock these down before we can make major progress in construction. The major roadblocks are mechanisms to rotate the entire robot and a claw strong enough to clamp on to the window and a claw that is strong enough. We plan to take these challenges head on. See plans.

Plans

We would like to test out 2 experimental solutions to the challenges above next week. For the arm, we are investigating a high reduction, but heavy, gearbox driving both pivot joints. This would involved having a gear that can move down a shaft while transferring power rotationally. We would like to build a prototype of this to make sure we can design the rest of the robot around it.

Stephanie is also planning on building a model of the claw out of laser cut materials to prototype the linkages. We need to do this to ensure that the claw can enact the close force we require.

Appendix A

Our firmware on the Arduino microcontroller.

```
#include <PID_v1.h>

#include <Encoder.h>

#include <Servo.h>

// hardware configuration
#define INPUT_POT A0
#define INPUT_RANGE A4
#define INPUT_BEND A1
#define INPUT_LIMIT 4
#define OUTPUT_DEBUG 13
#define OUTPUT_SERVO 9
#define OUTPUT_MOTOR_FORWARD 5
#define OUTPUT_MOTOR_BACKWARD 6
#define OUTPUT_STEPPER_STEP 7
#define OUTPUT_STEPPER_DIR 8
#define OUTPUT_STEPPER_EN 13
#define INPUT_ENCODER_A 2  //should be on an interrupt pin
#define INPUT_ENCODER_B 3  //should be on an interrupt pin
#define KDPP 0.9
#define KDPI 0.2
#define KDPD 0
#define KDVP 1
#define KDVI 0
#define KDVD 0

#define ENCODER_DT_MIN_MS 50
#define STEPPER_DELAY_US 2000
#define FILTER_SIZE 32

struct FilterData {
  unsigned int total;
  unsigned int index;
  unsigned int data[FILTER_SIZE];
};

// zero out all the filter parameters and data
void filter_init(struct FilterData *filter_data) {
  filter_data->total = 0;
  filter_data->index = 0;
  for(unsigned int i = 0; i < FILTER_SIZE; i++) {
    (filter_data->data)[i] = 0;
  }
}

// add
unsigned int filter_add(struct FilterData *filter_data, unsigned int new_value) {
  filter_data->total -= filter_data->data[filter_data->index];
  (filter_data->data)[filter_data->index] = new_value;
  filter_data->total += new_value;

  filter_data->index++;
  if(filter_data->index >= FILTER_SIZE) {
    filter_data->index = 0;
  }

  return (filter_data->total)/FILTER_SIZE;
}

// global data
// sensor readings
int encoder_position = 0;
int encoder_position_old = 0;
double encoder_velocity = 0;
unsigned int pot_value = 0;
unsigned int range_value = 0;
unsigned int bend_value = 0;
uint8_t limit_value_old = 0;
uint8_t limit_value = 0;
//gui commands
unsigned int gui_servo_setPoint = 0;
int gui_dc_position = 0;
int gui_dc_velocity = 0;
int gui_stpper_position =0;
// setpoints
unsigned int stepper_position = 0; //TODO should this be an unsigned int
unsigned int stepper_setpoint = 0;
uint8_t servo_setpoint = 0;
double motor_setpoint;
uint8_t motor_pwm_setpoint = 0;
uint8_t motor_pwm_direction = 0; // 0 = forwards
// program mode 0 = sensor, 1 = gui, velocity, 2 = gui, position, 3 = sensor, position
uint8_t program_mode = 0;
// other
char mode = 0;
byte index = 0;
char read_value[5];
unsigned int data_timer = 0;
uint8_t incomingByte = 0; // for incoming serial data
```

```
unsigned long time_old = 0;
unsigned long time_now = 0;
unsigned long time_dt = 0; // time between encoder updates
// PID
double pid_position_input;
double pid_position_output;
double pid_position_setpoint;
double pid_velocity_input;
double pid_velocity_output;
double pid_velocity_setpoint;

PID dc_position_PID(&pid_position_input,
                    &pid_position_output,
                    &pid_position_setpoint,
                    KDPP,KDPI,KDPD,DIRECT);
PID dc_velocity_PID(&pid_velocity_input,
                    &pid_velocity_output,
                    &pid_velocity_setpoint,
                    KDVP,KDVI,KDVD,DIRECT);
Servo servo;
Encoder kencoder(INPUT_ENCODER_A,INPUT_ENCODER_B);
struct FilterData bend_filter;
struct FilterData range_filter;



/*-------------------------------------------*/
/* Initializization code (run once via call from Arduino framework) */
void setup() {
  // establish direction of pins we are using to drive LEDs
  pinMode(INPUT_POT, INPUT);
  //pinMode(BendBefore, OUTPUT);
  pinMode(INPUT_RANGE, INPUT);
  pinMode(INPUT_BEND, INPUT);
  pinMode(INPUT_LIMIT, INPUT_PULLUP);
  pinMode(OUTPUT_DEBUG, OUTPUT);

  // output setup
  servo.attach(OUTPUT_SERVO); // servo
  pinMode(OUTPUT_MOTOR_FORWARD, OUTPUT);
  analogWrite(OUTPUT_MOTOR_FORWARD, 0);
  pinMode(OUTPUT_MOTOR_BACKWARD, OUTPUT);
  digitalWrite(OUTPUT_MOTOR_BACKWARD, 0);
  pinMode(OUTPUT_STEPPER_STEP, OUTPUT);
  digitalWrite(OUTPUT_STEPPER_STEP, 0);
  pinMode(OUTPUT_STEPPER_DIR, OUTPUT);
  digitalWrite(OUTPUT_STEPPER_DIR, 0);
  pinMode(OUTPUT_STEPPER_EN, OUTPUT);
  digitalWrite(OUTPUT_STEPPER_EN, 1);

  // filter setup
  filter_init(&range_filter);
  filter_init(&bend_filter);

  // PID setup
  dc_position_PID.SetOutputLimits(-255,255);

  Serial.begin(9600);
}


void step() {
  // enable controller
  digitalWrite(OUTPUT_STEPPER_EN, 0);
  delayMicroseconds(STEPPER_DELAY_US/2);
  // pulse up
  digitalWrite(OUTPUT_STEPPER_STEP, HIGH);
  delayMicroseconds(10);
  // pulse down
  digitalWrite(OUTPUT_STEPPER_STEP, LOW);
  delayMicroseconds(STEPPER_DELAY_US/2);
  // disable controller
  digitalWrite(OUTPUT_STEPPER_EN, 1);
}

void output_serial_data() {
  Serial.print(pot_value);
  Serial.print(" ");
  Serial.print(range_value);
  Serial.print(" ");
  Serial.print(bend_value);
  Serial.print(" ");
  Serial.print(program_mode);
  Serial.print(" ");
  Serial.print(encoder_position);
  Serial.print(" ");
  Serial.print(encoder_velocity);
  Serial.print(" ");
  Serial.print(time_dt);
  Serial.print(" ");
  Serial.print(pid_position_input);
  Serial.print(" ");
  Serial.print(pid_position_output);
  Serial.print(" ");
  Serial.print(pid_position_setpoint);
  Serial.println();
```

```
}


uint8_t bend_value_shift(unsigned int value_in) {
  if(value_in < 600) {
    return 0;
  } else if (value_in > 780) {
    return 180;
  } else {
    return value_in-600;
  }
}



/* Main routine (called repeated by from the Arduino framework) */
void loop() {
  data_timer++;

  // read sensor data
  if(millis() - time_now > ENCODER_DT_MIN_MS) {
    // time
    time_old = time_now;
    time_now = millis();
    time_dt = time_now - time_old;
    // encoder update on slow cycle
    encoder_position_old = encoder_position;
    encoder_position = kencoder.read();
    encoder_velocity = ((double)(encoder_position - encoder_position_old))/((double)time_dt);
  }
  // analog sensors
  pot_value = analogRead(INPUT_POT);
  range_value = filter_add(&range_filter, analogRead(INPUT_RANGE));
  bend_value = filter_add(&bend_filter, analogRead(INPUT_BEND));
  limit_value_old = limit_value;
  limit_value = digitalRead(INPUT_LIMIT);

  //checks to see if data has been sent
  if (Serial.available() > 0){
       //read the incoming byte:
      incomingByte = Serial.read();
      switch (incomingByte){
       case 'S':
         //servo mode
         index = 0;
         mode = 0;
         break;
```

```
     case 'R':
       //reset
       index = 0;
       mode = 1;
       gui_servo_setPoint = 0;
       gui_dc_position = 0;
       gui_dc_velocity = 0;
       gui_stpper_position = 0;
       break;
     case 'P':
       //Position DC
       index = 0;
       mode = 2;
       break;
     case 'V':
       //Velocity DC
       index = 0;
       mode = 3;
       break;
     case 'A':
      //Stepper position
       index = 0;
       mode = 4;
      break;
     default:
      read_value[index] = incomingByte;
      index ++;
       int sum;
      if(index > 3){
        sum = atoi(read_value);
        index = 0;
        switch (mode){
          case 0:
            gui_servo_setPoint = sum;
            break;
          case 2:
            gui_dc_position = sum;
            break;
          case 3:
            gui_dc_velocity = sum;
            break;
          case  4:
            gui_stpper_position = sum;
            break;
        }
      }
   }
}
```

```
// change program mode, if necessary
if(limit_value_old == 0 && limit_value == 1) {
  if(program_mode < 3) {
    program_mode++;
  } else {
    program_mode = 0;
  }
}



// determine proper control settings
if(program_mode == 1) {
  // gui control velocity
  servo_setpoint = gui_servo_setPoint;
  pid_velocity_input = encoder_velocity * 512;
  pid_velocity_setpoint = (double)(gui_dc_velocity);
  if(dc_velocity_PID.GetMode() == MANUAL) {
    dc_position_PID.SetMode(MANUAL);
    dc_velocity_PID.SetMode(AUTOMATIC);
  }
  dc_velocity_PID.Compute();
  motor_setpoint = pid_velocity_output;

  stepper_setpoint = gui_stpper_position; // 0-1024
} else if (program_mode == 2) {
  // gui control position
 pid_position_input = encoder_position;
 pid_position_setpoint = (double)(gui_dc_position);
 dc_position_PID.Compute();

  // process output
  motor_setpoint = pid_position_output;

  stepper_setpoint = gui_stpper_position; // 0-1024

  servo_setpoint = gui_servo_setPoint;
  if(dc_position_PID.GetMode() == MANUAL) {
    dc_velocity_PID.SetMode(MANUAL);
    dc_position_PID.SetMode(AUTOMATIC);
  }
  // todo
} else if (program_mode == 3) {
  // sensor control with motor position
  pid_position_input = encoder_position;
```

```
    pid_position_setpoint = (double)(pot_value);
    if(dc_position_PID.GetMode() == MANUAL) {
      dc_velocity_PID.SetMode(MANUAL);
      dc_position_PID.SetMode(AUTOMATIC);
    }
    dc_position_PID.Compute();

    // process output
    motor_setpoint = pid_position_output;

    stepper_setpoint = range_value; // 0-1024
    servo_setpoint = bend_value_shift(bend_value);;
  } else {
    // user motor velocity pid
    pid_velocity_input = encoder_velocity * 512;
    pid_velocity_setpoint = (double)(pot_value/4);
    if(dc_velocity_PID.GetMode() == MANUAL) {
      dc_position_PID.SetMode(MANUAL);
      dc_velocity_PID.SetMode(AUTOMATIC);
    }
    dc_velocity_PID.Compute();
    motor_setpoint = pid_velocity_output;

    stepper_setpoint = range_value; // 0-1024
    servo_setpoint = bend_value_shift(bend_value);
  }

  // output settings
  // set servo
  servo.write(servo_setpoint);

  // set motor output based on pot value
  if(motor_setpoint < 0) {
      motor_pwm_direction = 1;
      motor_pwm_setpoint = -(uint8_t)motor_setpoint;
    } else {
      motor_pwm_direction = 0;
      motor_pwm_setpoint = (uint8_t)motor_setpoint;
    }
  if (motor_pwm_setpoint < 16) {
    // have a dead-band for low values
    analogWrite(OUTPUT_MOTOR_FORWARD, 0);
    analogWrite(OUTPUT_MOTOR_BACKWARD, 0);
  } else if (motor_pwm_direction == 0) {
    // move forwards
    analogWrite(OUTPUT_MOTOR_FORWARD, motor_pwm_setpoint);
    analogWrite(OUTPUT_MOTOR_BACKWARD, 0);
  } else {
    analogWrite(OUTPUT_MOTOR_FORWARD, 0);
    analogWrite(OUTPUT_MOTOR_BACKWARD, motor_pwm_setpoint);
  }


  // set stepper motor to position based on bend sensor
  int stepper_position_error = stepper_setpoint - stepper_position;
  if (stepper_position_error > 8) {
    // set direction
    digitalWrite(OUTPUT_STEPPER_DIR, HIGH);
    step();
    stepper_position++;
  } else if (stepper_position_error < -8) {
    digitalWrite(OUTPUT_STEPPER_DIR, LOW);
    step();
    stepper_position--;
  }

  // output relevant data
  if (data_timer == 200) {
    data_timer = 0;
    output_serial_data();
  }
}
```