Sensors and Motors Lab

Individual Lab Report

Erik Sjoberg

Team C – Column Robotics

Rohan Thakker, Job Bedford, Cole Gulino

IRL 1

October 16, 2015

Individual Progress

Implemented PID DC motor control (position, velocity) via an IR distance sensor

I implemented the complete chain of functionality which enabled the position and velocity control of a DC motor via an IR distance sensor. The work consisted of:

1. Hooking up the Solarbotics L298 motor controller circuit
    a. Reading datasheets, debugging problems (multiple data sheet versions)
2. Interfacing with quadrature encoder via pin-change interrupts
3. Calculating a smoothed velocity measurement from encoder values
4. Writing a simple PID control loop which is (relatively) insensitive to irregular timings
    a. Decided against interrupt-based implementation for simplicity's sake
5. Interfacing with the SHARP 2Y0A02 IR sensor to acquire an analog distance measure
6. Smoothing and transforming raw values from the IR sensor
7. Tuning the controller for both speed and velocity control
    a. Motor state input from encoder position or calculated velocity
    b. Setpoint input from transformed distance measurement from IR sensor
8. Debugging and integrating code with the rest of the project state machine

After working out the bugs of the system, PID control of the DC motor was quite successful at smoothly reaching the desired velocity or position. It was a good learning opportunity to implement the controller myself, as opposed to leveraging the available PID control packages.

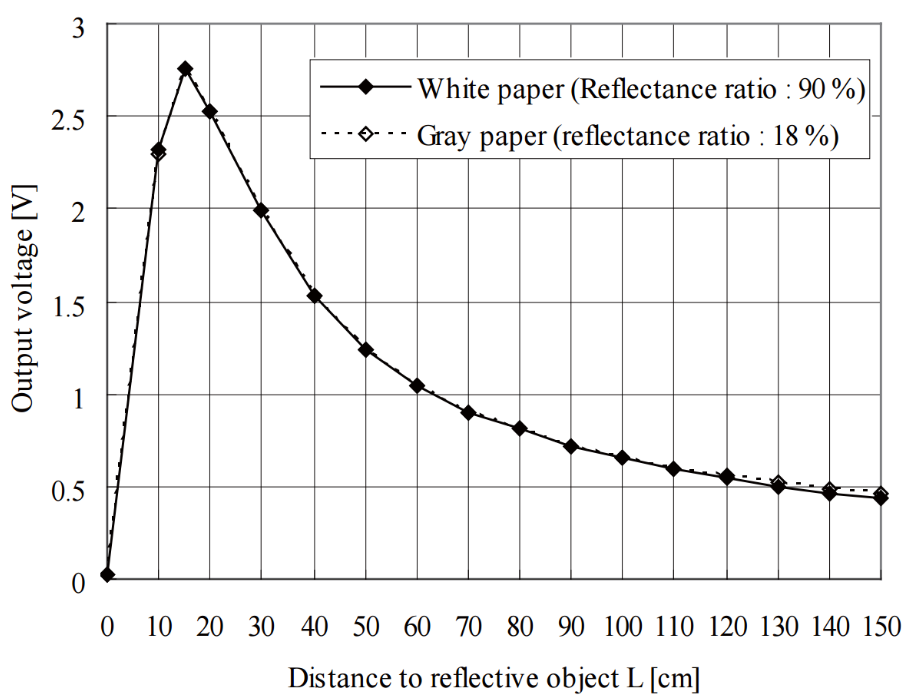

Figure 1: Transfer function of SHARP 2Y0A02

New Scrum management processes and tools researched, developed, and implemented

As our team's project manager, I've driven the continued iteration of our management processes and tools. The initial iteration of our "agile" planning process resulted in a large backlog of user stories (chunks of functionality) which lacked enough organization to be useful. This week I developed a new framework which organizes our deliverables (chunks of functionality) around the specific demos we plan to give in the fall.

A dramatic increase in clarity and utility was achieved by organizing the new deliverable/task tracking framework around the specific demos we will be performing in December. Since all table rows now map to individual tasks, progress and remaining work can be tracked effectively. The increased visibility of a google sheet that all teammates can access and update (notably absent from MS Project) is another massive advantage of this new management framework which we will use to organize our releases and sprints going forwards.

	WBS	Item Type	Task Name	Work	Customer Need
39	2	User Story	Epic: Search and Identify	179 hrs	High
40	2.1	User Story	Low-level control of drone / takeoff via ROS (AR drone)	32 hrs	High
41	2.1.1	Task	Write node to acquire drone data	7 hrs	
42	2.1.2	Task	Document and share MOVER / READER interface:	8 hrs	
43	2.1.3	Task	design and write MOVER node to issue command	14 hrs	
44	2.1.4	Task	Write test script to show control	1 hr	
45	2.1.5	Task	Add takeoff / land / abort features to MOVER	2 hrs	
46	2.2	User Story	Plan + execute next movement (lawnmower)	10 hrs	High
47	2.2.1	Task	Research robot_localization package	2 hrs	
48	2.2.2	Task	Research planning frameworks (movelet, ompl)	2 hrs	
49	2.2.3	Task	Implement planner (commands MOVER)	6 hrs	
50	2.3	User Story	Estimate current position in environment (localize)	0 hrs	High
51	2.3.1	Task	Research robot_localization package	0 hrs	
52	2.3.2	Task	Write state-estimator node	0 hrs	
54	2.4	User Story	Detect + avoid walls / stationary obstacles	35 hrs	Medium
55	2.4.1	User Story	Detect obstacles	15 hrs	Medium
56	2.4.2	User Story	Plan path around obstacles	20 hrs	Medium
57	2.5	User Story	Control (maintain) pose automatically (8X)	50 hrs	High
58	2.5.1	User Story	Track changes in pose	30 hrs	High
59	2.5.1.1	User Story	Detect distance from floor	10 hrs	High
60	2.5.1.2	User Story	Detect rotation/translation	20 hrs	High
61	2.5.2	User Story	Track changes in pose w/ low visibility	20 hrs	Low
62	2.6	User Story	Detect environment features	5 hrs	Medium
63	2.7	User Story	Display drone heartbeat signal	5 hrs	Medium
64	2.8	User Story	Estimate current position w/ low visibility	20 hrs	Low
65	2.9	User Story	Identify wellhead	22 hrs	High
73	3	User Story	Epic: Autonomously maneuver to pre-dock	32 hrs	High
74	3.1	User Story	Status update to user for "At Wellhead"	2 hrs	Medium
75	3.2	User Story	Avoid contact with wellhead	20 hrs	Medium
76	3.2.1	User Story	Detect wellhead structure	10 hrs	Medium
77	3.2.2	User Story	Plan path around wellhead to find dock	10 hrs	Low
78	3.3	User Story	Steadily hold position above dock	5 hrs	Medium
79	3.4	User Story	Orient appropriately for docking	5 hrs	Medium
80	4	User Story	Epic: Dock at wellhead	141 hrs	High
86	4.2	User Story	Status update to user for "Docking"	4 hrs	Medium
87	4.3	User Story	Manual docking at wellhead (prototype)	30 hrs	Medium
88	4.4	User Story	Automated docking at wellhead	60 hrs	High
89	4.4.1	User Story	Controlled approach to dock	30 hrs	High
90	4.4.2	User Story	Detect abnormal/failed docking attempt	10 hrs	Medium
91	4.4.3	User Story	Return to pre-dock position	20 hrs	Low
92	4.5	User Story	Detect successful docking	5 hrs	Medium
93	4.6	User Story	Rigidly lock to dock	20 hrs	Medium
94	4.7	User Story	Make electrical connection with dock	10 hrs	Low
95	4.8	User Story	Take image (post-docking)	4 hrs	Low
96	4.9	User Story	Transmit image of dock	2 hrs	Low
97	5	User Story	Epic: Fully integrated system	0 hrs	High

1	No.	Demo Functionality	Tasks	Owner	Sprint	Status	Est. Work	Remaining
2	1	Position tracking	Demonstration of optical odometry with april tag matt as global reference. Manual control of drone.					
3	1.1	Display global position reference	Research alternative global reference strategies			Not Star	4	4
4			Research and choose best localization tag type			N	2	2
5			Create mat / tile of tags			N	4	4
6			Extract position of tags from video feed			n	4	4
7			Calibrate & verify global position estimate			N	4	4
8			Calculate + publish global position based on tag positions			N	2	2
9	1.2	Display calculated global position estimate	Research exsisting methods of visual odometry and sensor fusion			N	4	4
10			Research + document robot_localization package	Erik	1	N	2	2
11			Implement 2d x,y map display in ROS	Erik	1	N	4	4
12			Test various algorithms on existing datasets			N	8	8
13			Implement algorithm on pixhawk or PC/SBC			N	16	16
14			Evaluate estimate against global position			N	8	8
15	1.3	Aquire sensor information from camera and position sensors	Figure out how to communicate with AR.Drone			N	2	2
16			Write READER node for AR.Drone			N	4	4
17			Write READER node for Iris+			N	8	8
18	2	Hardware setup on Iris	Demonstrate camera feed, data feed, SBC talking to Pixhawk					
19	2.1	Mount sensors and SBC on IRIS	Decide on sensors and purchase			N	8	8
20			Design mounting hardware			N	4	4
21			Fabricate and assemble			N	8	8
22	2.2	Develop electronics for power and interfacing	Evaluate onboard pixhawk power supply			N	2	2
23			Wire up power for SBC / Sensors.			N	4	4
24	2.3	Develop communication interface	Setup serial communication between sensors and processor			N	8	8
25			Setup communication protocols between pixhawk and SBC			N	8	8
26			Setup manual control / auto control switching			N	4	4
27			Gather video feed			N	4	4
28	3	Prototype of dock	Demonstrate one proof of concept, one actual prototype					
29	3.1	Manual docking at wellhead (iris+)	Develop safety protocols			N	2	2
30			Test manual docking			N	4	4
31	3.2	Prototype of dock sub-system	Iterate dock concepts / design			N	4	4
32			Design, fabricate, build, develop rough prototype.			N	8	8
33			3D print needed parts			N	4	4
34			Manually fabricate needed parts			N	4	4
35	4	Open-loop ARDrone Control	Demonstrate takeoff, move, land at push of ROS button					
36	4.1	Display ROS node graph	Setup launch script for nodes and topics			N	2	2
37	4.2	Low-level open-loop control of drone / takeoff via ROS (AR drone)	Set up ROS framework / GIT repo	Erik	1	Done	4	0
38			Write node to acquire AR.Drone data	Rohan	1	Started	8	6
39			Document and share MOVER / READER interfaces			N	2	2
40			Design and write MOVER node to issue commands	Job	1	N	8	8
41			Write test script to show control			N	4	4
42			Add takeoff / land / abort features to MOVER			N	4	4
43	5	Non-Demo Focus Areas						
44	5.1	Low-level open-loop control of Iris+	Write node to acquire Iris data			N	16	16
45			Modify MOVER / READER interfaces			N	4	4
46			Write test script to show control			N	8	8
47	5.1	Automated takeoff / land Iris+	Ensure stability			N	16	16
48			Ensure gradual landing			N	16	16

Figure 2: Scrum management eye-chart: Function-focused (left) vs. Demo-focused (right)

Challenges

Solarbotics PWM pin datasheet version inconsistency

When implementing the PWM control for the DC motor, I experienced issues with PWM control in one direction. It turns out that there are two versions of the datasheet linked on the solarbotics homepage, one under “datasheet” and the other under “documentation”. Both of these are the same document, where the “datasheet” document is the 2008 revision: https://solarbotics.com/download.php?file=43 and the “documentation” is the 2010 revision: https://solarbotics.com/download.php?file=40.

It turns out I had implemented the circuit using the incorrect version of the datasheet (PWM on the ENABLE pin is the correct approach for the hardware I was given). Unfortunately, the circuit still worked to some extent, even with the wrong logic table but showed a very jittery behavior under reverse PWM control. To avoid this type of issue in the future I intend to confirm the waveforms for each input/output on my circuits behave as expected before moving to the next steps.

Enable	L1	L2	Result
L	L	L	OFF
L	L	H	OFF
L	H	L	OFF
L	H	H	OFF
H	L	L	BRAKE
H	L	H	FORWARD
H	H	L	BACKWARD
H	H	H	BRAKE
H	L	L	BRAKE
H	PWM	H	FWD-SPD
H	PWM	L	BCK-SPD
H	H	H	BRAKE

ENABLE	L1	L2	Result
L	L	L	OFF
L	L	H	OFF
L	H	L	OFF
L	H	H	OFF
H	L	L	BRAKE
H	L	H	FORWARD
H	H	L	BACKWARD
H	H	H	BRAKE
PWM	L	L	PULSE-BRK
PWM	L	H	FWD-SPD
PWM	H	L	BCK-SPD
PWM	H	H	PULSE-BRK

Figure 3: Solarbotics L298 Logic Table 2010 revision (left) vs 2008 revision (right)

Teamwork

The team has been working well together, and I could not ask for better teammates.

- Cole: Quite busy with Machine Learning class, but participating well and motivated.
- Job: Not a fan of meetings, but pulls his weight when working on systems.
- Rohan: Has a tendency to return re-evaluate decisions after they’ve been made, but this has worked to the team’s benefit more often than not. Incredibly capable and a definitely the most experienced roboticist on our team.

We have up until this point been suffering from a lack of organization around deliverables and due dates for course assignments due in large part to the lack of a consistent tracking / assignment method for tasks which is accessible to all team members. I believe that the latest iteration of our management framework (discussed above) will remedy this lack of organization.

Plans for Upcoming Work

For the coming week, I plan to do the following:

- Research and document tools in the robot_localization package
- Implement a 2d x,y map in ROS to display estimated position
- Continue refining WBS and management tool usage patterns

Appendix A: DC Motor Control Code

```
#include <pins_arduino.h>

// Define pin numbers
#define dcMotorEnablePin 6
#define dcMotorPinA 11
#define dcMotorPinB 8
#define encoderPinA 4
#define encoderPinB 5
#define opticalPin A1


// Variables
volatile int state;
long debounceDelay;
long opticalSensorVoltage;
long opticalSensorVoltageSmooth;
int dcMotorAngle;


//DC encoder
volatile int encoderPos = 0;
int lastEncoderPinA = LOW;
int lastEncoderPinB = LOW;
int lastEncoderPos = 0;
```

```
double lastSpeedTime = 0;
int epA = LOW;
int epB = LOW;
double dcSpeedMeasured; //Degrees per second
double dcSpeedSmoothed;
double dcTargetSpeed; //Degrees per second

//PID: Reference http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-
introduction/
//Define Variables we'll be connecting to
/*working variables*/
unsigned long lastTime;
double Input, Output, Setpoint;
double errSum, lastErr;
double kp, ki, kd;
double avgSpeed = 0;
double avgOutput = 0;

// pins_arduino Reference http://playground.arduino.cc/Main/PinChangeInterrupt
void pciSetup(byte pin){
    *digitalPinToPCMSK(pin) |= bit (digitalPinToPCMSKbit(pin));  // enable pin
    PCIFR  |= bit (digitalPinToPCICRbit(pin)); // clear any outstanding interrupt
    PCICR  |= bit (digitalPinToPCICRbit(pin)); // enable interrupt for the group
}

ISR (PCINT2_vect){  // handle pin change interrupt for D0 to D7 here
   epA = digitalRead(encoderPinA);
   if (epA != lastEncoderPinA) //Only trigger if pin A has changed = 360 counts per revolution
   {
     if (epA == digitalRead(encoderPinB)) {
       encoderPos++; //360 counts per revcolution clockwise
      } else {
       encoderPos--;
      }
      lastEncoderPinA = epA;
   }
}

void setup(){
  digitalWrite(dcMotorEnablePin, HIGH);

  //DC motor control
  pinMode(encoderPinA, INPUT);
  pinMode(encoderPinB, INPUT);
```

```
  digitalWrite(dcMotorPinA, LOW);
  digitalWrite(dcMotorPinB, LOW);
  pciSetup(encoderPinA);
  //PID
  Input, Output, Setpoint = 0;
  lastTime = 0;
  SetTunings(2, 0, 0);

  opticalSensorVoltage = 0;
  Serial.begin( 9600 );
  Serial.setTimeout(5);
}

void loop(){

  switch( state ){ // Potentiometer servo control
    case 1: {  // DC Motor Velocity Control
      // Sensing range ~ 50 (far) to 550 (close)
      opticalSensorVoltage = measureOpticalSensorVoltage();
      if(opticalSensorVoltage > 50){
        dcTargetSpeed = 2 * (300 - opticalSensorVoltage);
        dcSpeedMeasured = measureDCSpeed();
        dcDirect = (dcSpeedMeasured < 0)? 0 : 1;
        //Update PID values
        Input = dcSpeedMeasured;
        Setpoint = dcTargetSpeed;
        Compute(); //Modifies Output variable
        moveDCMotor();
      }
      else{
        digitalWrite(dcMotorPinA, LOW);
        digitalWrite(dcMotorPinB, LOW);
        lastTime = millis();
      }
      break;
    }
    case 2: {  // DC Motor Position Control
      // Read sensor voltage
      opticalSensorVoltage = measureOpticalSensorVoltage(); // int( 50-550 )
      // Set PID input and setpoint
      // Setpoint is the desired state
      Setpoint = (guiCntrl) ? dcMotorAngle : opticalSensorVoltage; // Target state in degrees
      Input = encoderPos; // Actual encoder degree value
      // Compute direction we need to go
```

```
      Compute(); // Modifies Output Global variable
      moveDCMotor(); // Run DC motor control
      break;
    }

    if (state != 1 && state != 2){
      lastTime = millis(); //Prevent PID windup
      digitalWrite(dcMotorPinA, LOW); // Switch off the DC Motor
      digitalWrite(dcMotorPinB, LOW);
    }
  }
  delay(15);
}

void Compute(){
   /*How long since we last calculated*/
   unsigned long now = millis();
   double timeChange = (double)(now - lastTime);

   /*Compute all the working error variables*/
   double error = Setpoint - Input;
   errSum += (error * timeChange / 1000);
   double dErr = (error - lastErr) / timeChange;
   /*Compute PID Output*/
   Output = kp * error + ki * errSum + kd * dErr;

   /*Remember some variables for next time*/
   lastErr = error;
   lastTime = now;
}

void SetTunings(double Kp, double Ki, double Kd){
   kp = Kp;
   ki = Ki;
   kd = Kd;
}

double measureDCSpeed(){
   double speedNow;
   double speedNowSmooth;
   long now = millis();
   if(now - lastSpeedTime > 0){
     speedNow = 1000 *  //Units in seconds
            (encoderPos - lastEncoderPos) /  // 360 encoders per rev
```

```
            (now - lastSpeedTime);
    lastEncoderPos = encoderPos;
    lastSpeedTime = now;

    dcSpeedSmoothed = (2 * dcSpeedSmoothed + speedNow) / 3;
  }
  return dcSpeedSmoothed;
}

long measureOpticalSensorVoltage(){
  long osv;
  osv = analogRead(opticalPin);
  //Smoothing
  opticalSensorVoltageSmooth = (3 * opticalSensorVoltageSmooth + osv) / 4;

  return opticalSensorVoltageSmooth;
}

void moveDCMotor(){ //Limit output
  if (Output > 255){
    Output = 255;
  }
  if (Output < -255){
    Output = -255;
  }
  if (Output >= 0){  //counter clockwise
    digitalWrite(dcMotorPinB, HIGH);
    //analogWrite(dcMotorPinA, Output);
    digitalWrite(dcMotorPinA, LOW);
    analogWrite(dcMotorEnablePin, Output);
  }
  else{ //clockwise
    double reverse = -Output;
    //Serial.print("\t rev: ");Serial.println(reverse);
    digitalWrite(dcMotorPinB, LOW);
    digitalWrite(dcMotorPinA, HIGH);
    analogWrite(dcMotorEnablePin, reverse);
  }
}
```