Omnidirectional Unmanned Manipulation Robot with Kinect Control

Progress Update 1

November 14, 2018

Summary

The primary goals for project update 1 that we detailed in the project proposal were to determine the compatibility and feasibility of our different hardwares and softwares, to order all the parts we would require for our robot, and to create models/diagrams detailing how the hardware and circuits would come together. Overall, we we were successful in achieving all these goals, and are confident that our project is both feasible and achievable. Though we have not had to make major changes to our project scope from what we initially envisioned, small changes here and there have helped guide our project in our desired direction.

1) Detailed Project Description

Mechanics

- We will be using the following components to assemble our robot:
- From TETRIX, 3 pieces of each:
    - DC Motor
    - Motor Mount
    - Motor Shaft Hub
    - 4'' Omni Wheels
- Balsa / Plywood that we can laser cut into shape
- MakeBlock Gripper
- Custom L-Bracket
- Arduino Uno
- Arduino Motor Shield

The following figures present a mockup of what our robot will look like, when assembled.

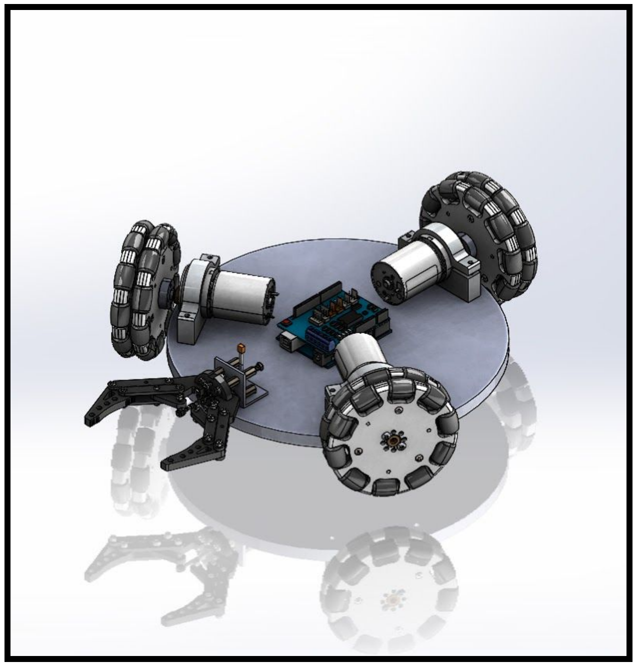
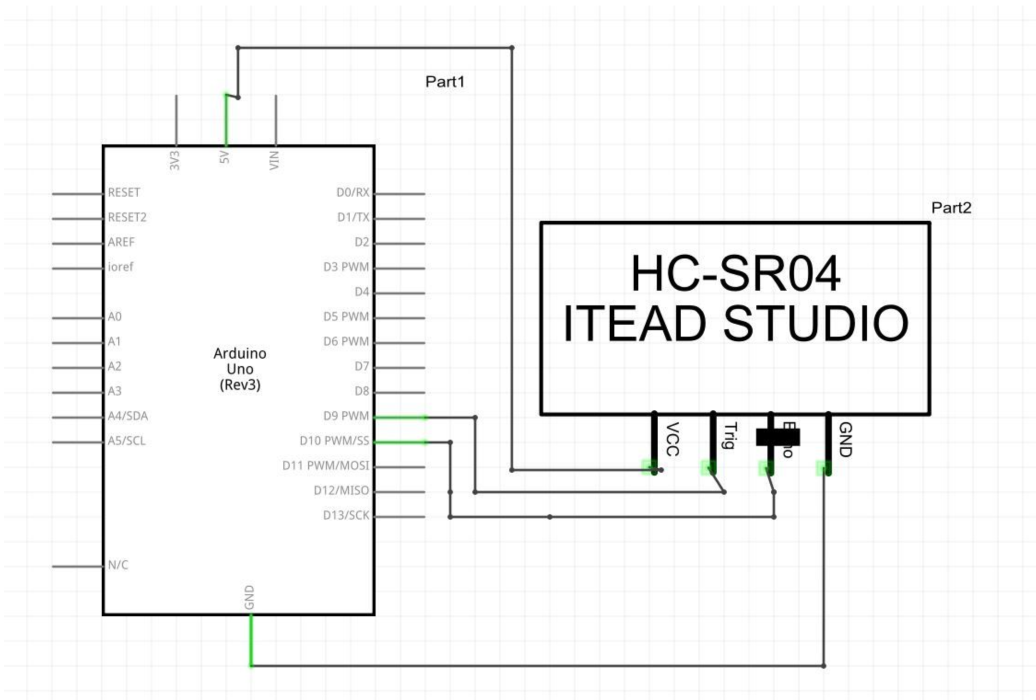
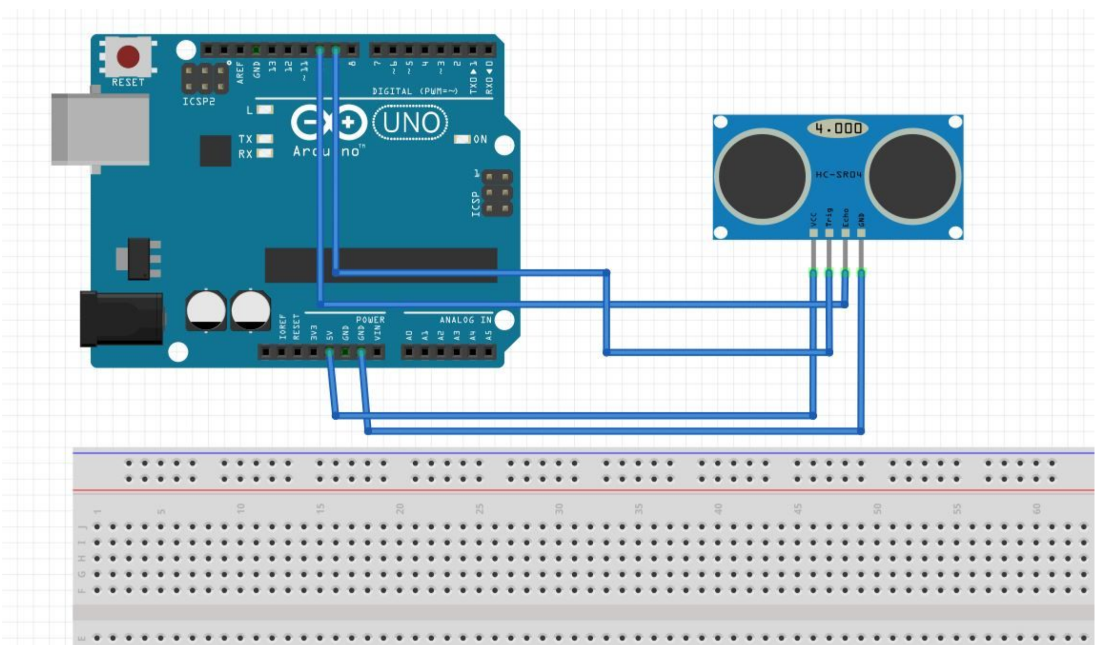


Figure 1: Isometric View of the Robot Assembled

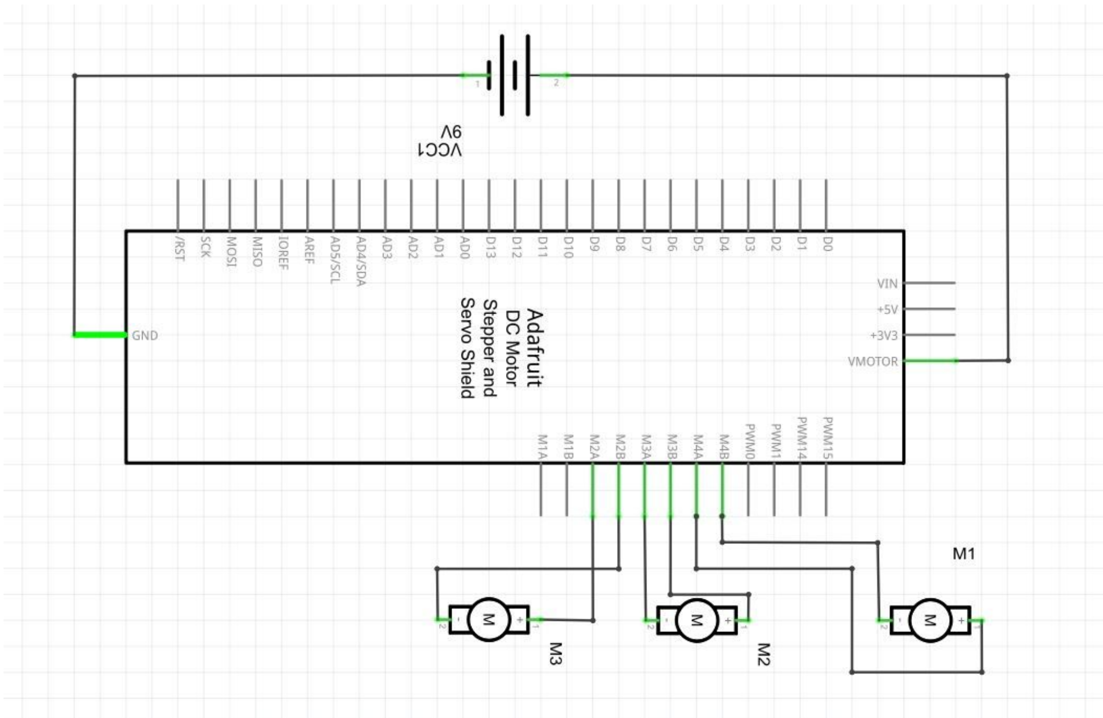
Figure 2: Isometric View of the Robot Assembled

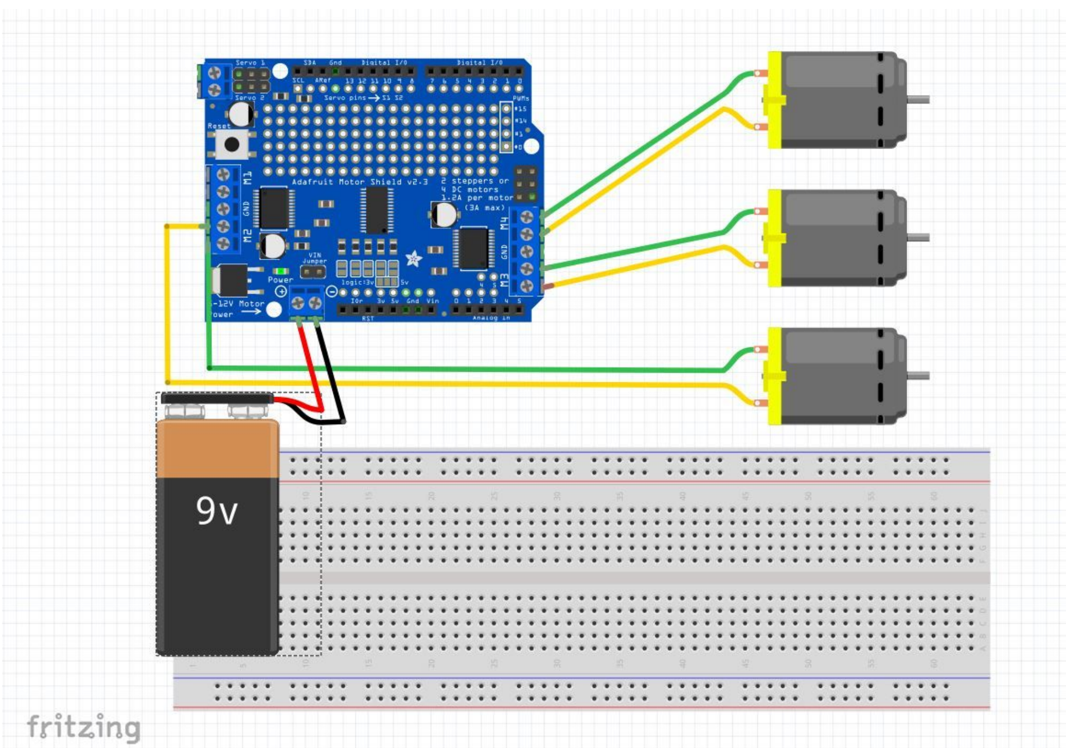
Electronics





We plan to include proximity sensors. We will use Elegoo HC-SR04 Ultrasonic sensor. We are planning to use a total of two sensors with an AND logic gate to perform a Top Secret™ task.





This is a simple circuit schematic we plan to use to power three wheels. We connect three servo motors to the motor shield, which will be powered by a 9V battery.

Code / Software

In this section, we will demonstrate the code we will use in this project. This sections includes two flow charts that demonstrate the general idea of the code. We also created an additional demo to show you how it really works.

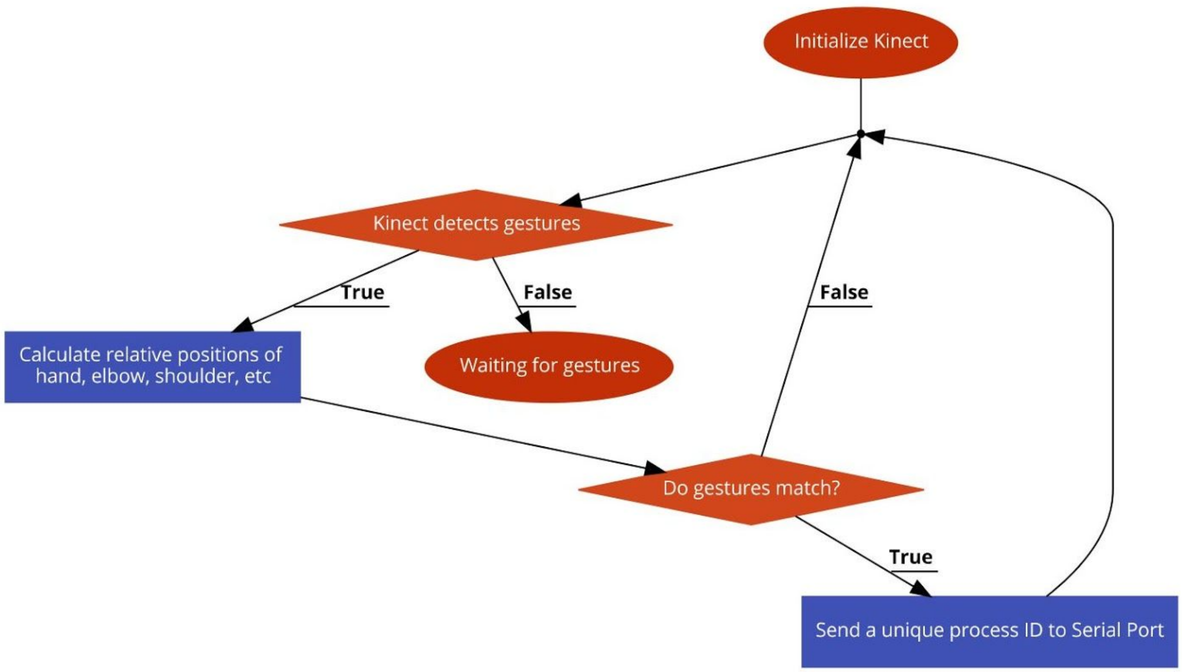


Figure 3: Kinect code flow chart

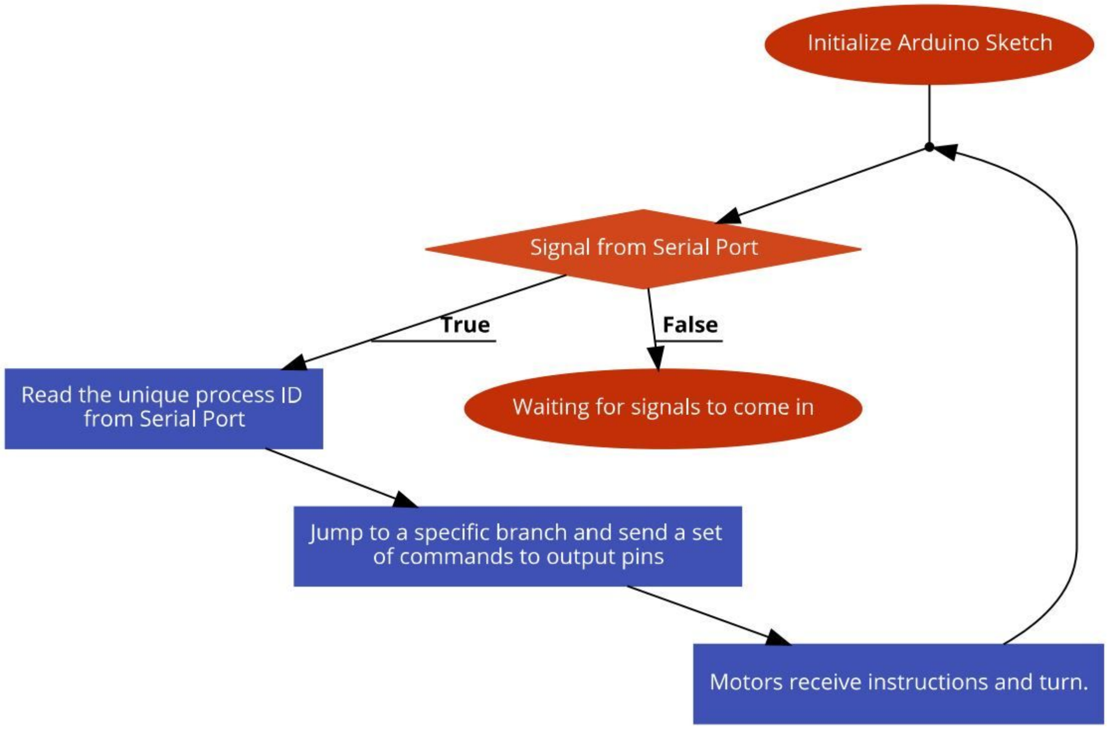


Figure 4: Arduino code flow chart

To better demonstrate how this code works, we prepared a short demo.

```
int ledPin = 2;

void setup() {
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
  digitalWrite(ledPin, LOW);
}

void loop() {
  if (Serial.available() > 0) { // signal deteced from Serial port
    char incomingByte = Serial.read();

    if (incomingByte == 'o') {
      digitalWrite(ledPin, HIGH);
    } else if (incomingByte == 'c'){
      digitalWrite(ledPin, LOW);
    } else { // unrecognized signal from Serial port
      for (int i=0; i<3; i++) {
        digitalWrite(ledPin, LOW);
        delay(500);
        digitalWrite(ledPin, HIGH);
        delay(500);
      }
      digitalWrite(ledPin, LOW);
    }

  }

}

```

This is the Arduino code demo. The loop running constantly and waiting for the signal from serial port. Once it detects signal, it checks whether the incoming signal is character ‘o’ (short for open) or character ‘c’ (short for close). If it’s ‘o’, it will write HIGH to output pin, which is connected to an LED. As a result, LED will light up. (Code could be found at: https://github.com/shuy98/24354TP/tree/master/progress_1)

Below is Kinect code demonstration. In this demo, I’m using left arm as an example. When I bend my arm and the angle is less than 90 degrees, it will send a unique ID (a character in this case) to serial port. When arduino receives that ID, the LED connected to Arduino will light up. Otherwise, it will send a different ID to serial port and Arduino will turn the LED off.

```python
def run(self):
    # -------- Main Program Loop -----------
    while not self._done:
        # --- Main event loop
        for event in pygame.event.get(): # User did something
            if event.type == pygame.QUIT: # If user clicked close
                self._done = True # Flag that we are done so we exit this loop

            elif event.type == pygame.VIDEORESIZE: # window resized
                self._screen = pygame.display.set_mode(event.dict['size'],
                                              pygame.HWSURFACE|pygame.DOUBLEBUF|
                                              pygame.RESIZABLE, 32)
```

This is the main loop that keeps the Kinect running. It will wait for gestures to come in.

```python
            if self._bodies is not None:
                for i in range(0, self._kinect.max_body_count):
                    body = self._bodies.bodies[i]
                    if not body.is_tracked:
                        continue
                    if body.is_tracked:
                        joints = body.joints

                        # left hand x and y position
                        if joints[PyKinectV2.JointType_HandLeft].TrackingState !=
                        PyKinectV2.TrackingState_NotTracked:
                            self.cur_left_hand_height = joints
                            [PyKinectV2.JointType_HandLeft].Position.y

                        if joints[PyKinectV2.JointType_HandLeft].TrackingState !=
                        PyKinectV2.TrackingState_NotTracked:
                            self.cur_left_hand_width = joints
                            [PyKinectV2.JointType_HandLeft].Position.x

                        self.position_left_hand = (self.cur_left_hand_width,
                        self.cur_left_hand_height)
```

When a gesture is detected, it stores the positions of the joints into variables.

```python
self.left_arm_angle = self.calc_angle(self.position_left_hand, self.position_left_elbow, self.position_left_shoulder)
```

This portion of the code packs the position data and sends them to a helper algorithm that will calculate the angle of the bent.

```python
# calculates the position of point_2 relative to point_1
def calc_position_rel(self, point_1, point_2):
    x_diff = point_2[0] - point_1[0]
    y_diff = point_2[1] - point_1[1]
    return (x_diff, y_diff) # returns a tuple

# calculates the distance between two points
def calc_distance(self, point_1, point_2):
    x_diff = point_2[0] - point_1[0]
    y_diff = point_2[1] - point_1[1]
    result = math.sqrt(x_diff**2 + y_diff**2)
    return result

# calcualtes the dot product between point_1 and point_2
def calc_dot_product(self, point_1, point_2):
    return point_1[0] * point_2[0] + point_1[1] * point_2[1]

# calculates angle(1_middle_2)
def calc_angle(self, point_1, point_middle, point_2):
    point_1_rel = self.calc_position_rel(point_middle, point_1)
    point_2_rel = self.calc_position_rel(point_middle, point_2)
    cos_angle = (self.calc_dot_product(point_1_rel, point_2_rel)/
                (self.calc_distance(point_middle, point_2) *
                 self.calc_distance(point_middle, point_1)))
    return math.acos(cos_angle) / math.pi * 180
```

This is the angle calculation algorithm. It utilizes the concept of dot product.

$$a \cdot b = |a||b|cos\theta$$

It uses the x and y positions of three recorded points (i.e. hand, elbow, and shoulder). Then it calculates the distance between each of them. By using the definition of dot product, we can get the angle of the elbow.

```python
if (self.left_arm_angle <= 90):
    msg = 'o'
    uno.write(msg.encode())
else:
    msg = 'c'
    uno.write(msg.encode())
```

Finally, it reads the angle of the elbow and sends the corresponding ID to serial port.

(Comprehensive code could be found at: https://github.com/shuy98/24354TP/blob/master/run.py)

We also did testing on both stepper motor and DC motor. Source file could be found at

https://github.com/shuy98/24354TP/tree/master/test_stepper

https://github.com/shuy98/24354TP/tree/master/DCMotorTest

Two video demos:

https://drive.google.com/open?id=10e9v8KFh-VoghbmWhZYgv0whkZy-m0J-

https://drive.google.com/open?id=1bp5BivpgAqbur-f5CCS3yjhSytn4Ww5T

Reflections

- We definitely learned about how to choose the right motor - we had a lot of group discussions about this and have developed better intuition on the topic.

- We're probably using up more of our budget than we want - if something breaks like a motor burning out or something, we don't have too much cushion with our remaining budget which is a risk that we are taking.

2) Issues Encountered

Most of the issues we encountered revolved around finding components of the robot which were compatible with each other. There were two major compatibility tests that we had to make. The first was to see if we could control a motor from a Kinect by interfacing through an Arduino. As displayed in the videos linked in Code/Software section, we were able to get things working by interacting with the Arduino through a Serial port. Our second major compatibility test was that we find a motor that could fulfill our expected torque requirements, but also be compatible with our wheels without us having to create any adaptors. This was an issue primarily because it took up a lot of time - we had to keep searching for new motors of different companies and setups to see what would fit our requirements the best. We ended up going with TETRIX motors, as they also sell adaptors for the motors to their omnidirectional wheels, and even have available a motor mount, making our lives a lot easier.

3) Proposed Changes to Project Scope

Overall, we haven't had to change our project scope too much. The biggest change is probably the fact that our gripper is no longer attached to our robot using a 4 bar linkage. This is because we ultimately deemed it an unnecessary complication. We will instead be creating a simple shape with styrofoam. This shape will be placed at a slight height, allowing the gripper to simply close around it and then have the robot move without dragging the object on the floor.