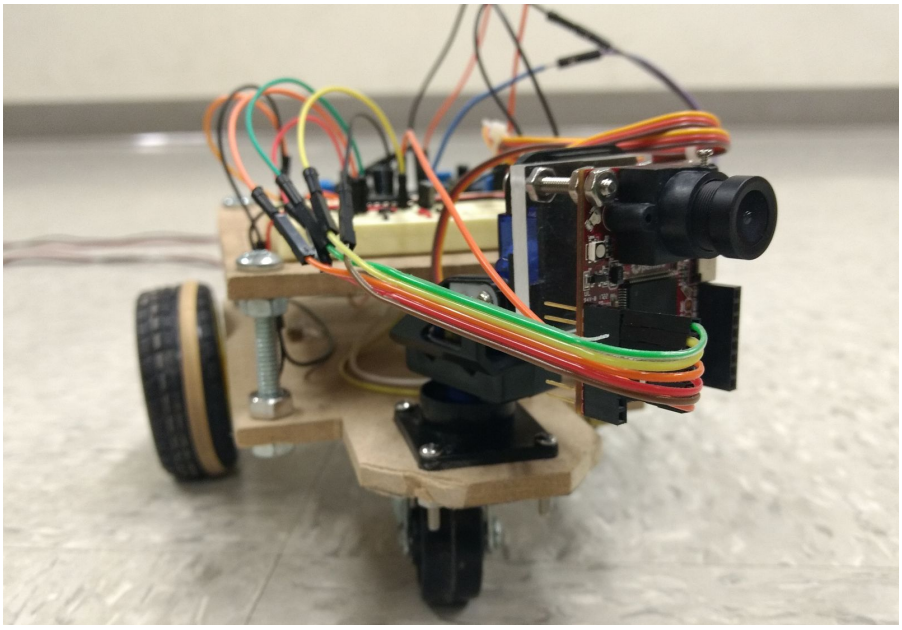
Machine Vision Robot (MVR): Final Report

Jansen Quiros (A12060650), Trevor McCleerey (A14234522)
PHYS 124 - Fall 2017



Motivation

The progression of the automobile has ever-favored consumer safety. First seat belts, followed by airbags; each year adding innovations to make the car driving experience safer. The next big progression in automobile safety is in vehicle autonomy. As such, we undertook a project that is a small-scale approach to the large-scale innovations currently being made in the industry. We shall investigate some of the key design principles in autonomous vehicular mobility. By designing a smart robot that is able to detect an object and navigate itself to it, we will familiarize ourselves with some of the engineering intuition involved in the major advancements being made with autonomous vehicles.

Functional Definition

The purpose of this project was to design and fabricate an autonomous ball-chasing robot. The functionality of the robot is coded in Python using the image processing unit's (OpenMV) integrated development environment (IDE). The robot is programmed to follow a specific colored object using the OpenMV camera and machine vision algorithms. A ball can be placed in front of the robot and it will recognize its color and relative position. When the ball is translated, the robot follows until the ball is at rest or out of range.

There are two sets of motors controlling the overall behavior of the robot. Two servo motors have the OpenMV camera mounted on them, and is programmed to center the object's position. The camera identifies a "blob" by drawing a rectangle around it. A *pan-servo* adjusts the camera horizontally to center the object in the x-direction and a *tilt-servo* will adjust the camera vertically to center the object in the y-direction. The servo motors use proportional control, adjusting in large increments if the object is far from the center of the image, and small increments when it is near the center of the image. A pair of DC motors are employed to drive the device and keep the ball within a tolerable range of operation. The DC motors are mounted to wheels which act as the means of both locomotion and steering. The camera receives the area of the object and compares it to a predefined object area tolerance and the DC motors move the robot to be within the tolerance range.

Assembly Components

The design of the machine vision robot requires the following:

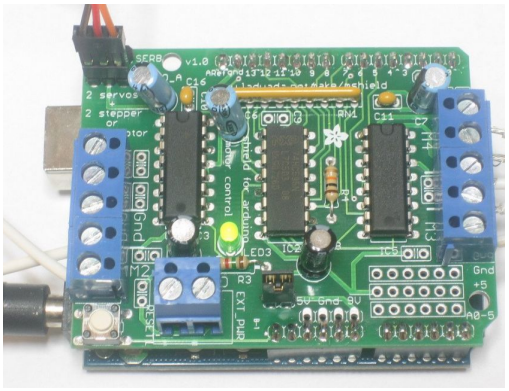
(1) OpenMV Cam M7

The OpenMV Cam M7 is a small, low-power microcontroller board which allows for easy application implementation using machine vision in real-world. Several of its applications include color tracking, frame differencing, and marker tracking.

This will be the most expensive unit on our project at a cost of $65. However, machine vision is widely applicable, and this camera should find use in future projects.

Motor Shield

Because the OpenMV is incapable of powering the servo motors, the servos will be connected to the motor shield. The PWM signal is sent from the OpenMV but the motor shield will have a separate power source for the servos.



(1) Mini Pan-Tilt Kit

The pan-tilt kit will equip the robot's image processor with a full range of motion using two micro servos. The pan-tilt can rotate roughly 180° and can tilt up and downwards around 150°. The micro servos assembled with the kit are SG-90 type and include a mount for the OpenMV Cam M7.

This unit may be reused in future projects.

(2) DC Motors

A standard DC Motor will be used to rotate the rear wheels of the robot. DC motors will be programmed to rotate forward and backward.

The DC motor may be reused for future projects.

(3) Wheels

The wheels will be made of a rubber material to maximize friction.
Two larger rear wheels will be mounted to DC motors and one swivel caster wheel will be placed in the front. Occasionally, we would wrap rubber bands around the wheels to have more traction because the lab floors are very slippery.

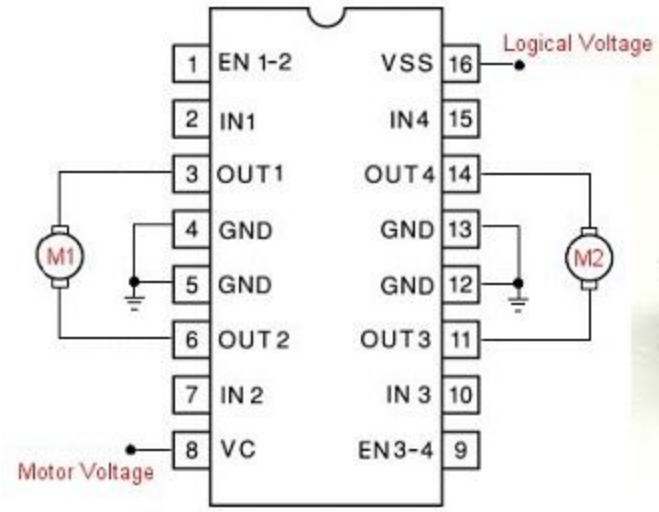
Power Supply

A 4-channel DC power supply will be used to deliver power to the motor shield (servos) and the DC motors. It is required that the DC motors and the servos are separately supplied because the OpenMV cannot support a shared supplied to the two subsystems.

Both channels will be supply 5V.

Breadboard Assembly

- Texas Instruments SN754410 - Quadruple High-Current Half-H Driver
  - Used to allow voltage to be applied across a load in the opposite direction
- 25 μF Capacitor
- Jumper Wires



Software Development

Machine Vision Algorithm

Machine Vision for following objects is easily attainable for following high contrasting colored objects. In our demonstration, we choose the color green for the OpenMV to follow. The program determines the green objects in front of the robot and draws a rectangular box on the largest object.

The image that is processed is essentially an array of "pixels" with an approximate size of (300, 220). The size of the camera's array was determined by having the camera stationary and have it return the coordinates of an object placed at the bottom right corner. Therefore, the center point of the boxed object should align to roughly (150, 110). The top left corner of the image represents (0,0) and the bottom right corner represents (~300, ~220).

The pan-servo and tilt-servo center the camera's view of the object horizontally and vertically, respectively.

The logic to adjust the center an object is simple: if the object is not centered, the horizontal and vertical difference between the center and the object's center point is computed to determine the angle of which the servos should be adjusted to.

Using basic control theory, we determined that the change in position and change in angle is proportional. From trial and error, it was determined that to center an object the ratio of the change in pixels to the change in angle on the Servo is 12:1.

$$\frac{\#\ of\ pixels\ from\ center}{Servo\ Angle} = \frac{12\ pixels}{1\ angle}$$

Camera detects the (largest) green object and draws a rectangle around it and a cross in the center.

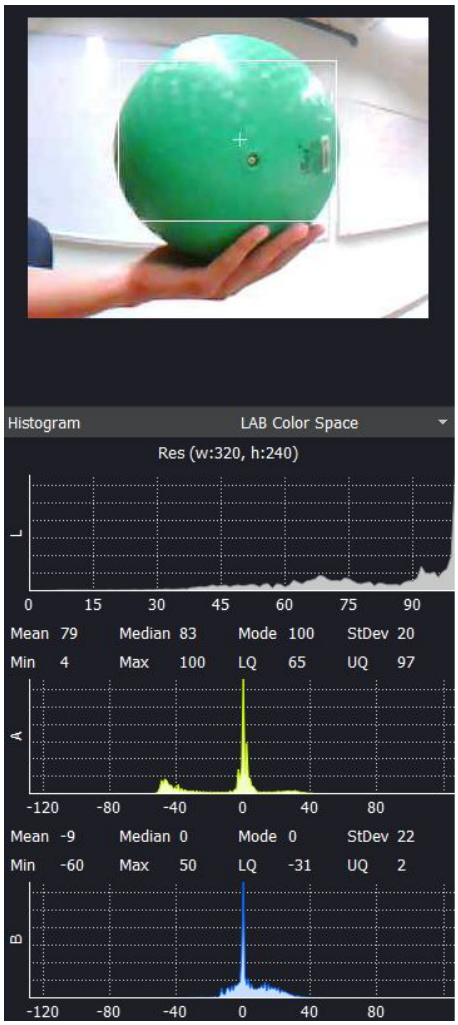
Locomotion Algorithm

Since the pan-tilt is limited to 180°, the robot must be able to move itself to follow an object that moves outside of this range. The DC motors can move the the robot forward, backwards, and turn left or right. The rectangle drawn around an object has a corresponding area that we can use to program the robot to manuever itself.

First, for a specific object we define a close range for the robot to be in the vicinity of the object so it has a reference area to work with. If the object moves away, the area becomes smaller and so the robot is programmed to run the DC motors forward. Likewise, if the object moves closer to the robot, the DC motors will operate backwards.

When the object is in an ideal range but the pan-servo motor is above 120° or below 60°, the robot will turn to adjust the focus of the ball.

Machine Vision Calibration



To calibrate the color threshold that we wanted the OpenMV to specifically follow, we tuned the LAB colorspace.

```
thresholds = [(75, 93, -44, -24, -3, 7),
              (54, 88, -52, -26, 0, 20),
              (71, 91, -51, -29, -6, 8)]
```

In our program, we defined three possible locations in the lab where the ball may be located at.

The *thresholds* array is of size N x 6, where N is the number of locations we define for our ball since different areas in the lab change how the balls threshold is defined. For example, the camera looking at the ball from one direction could have significant shadows that affect how the machine vision interprets the blob. Accounting for multiple thresholds allows a more robust blob detection platform.

The 6 columns define the (L_{min}, L_{max}, A_{min}, A_{max}, B_{min}, B_{max}). To the left we have an example of what these values are when the ball is placed around chest height with the whiteboard in the background. The LAB graphs shows a very broad range of values, so we take the peak value and a tolerance around it to acquire the min/max values.

We did multiple iterations on what would be the best set of values for our ball and implemented it on our code.

Electrical Considerations

Initially, we were unaware of the power limitations of the OpenMV camera. The OpenMV can only sink or source up to 25mA, while the DC motors can draw up to 1A in current, and the servos around 0.5A. This meant we would have to drive these two motor systems using motor drivers. We were able to power the servos and the camera but we needed to figure out a solution to power the DC motors.

Our final power design included the use of a 2 channel power supply, where 1 channel connected to the H-Bridge powering the DC motors with 5V and another channel connected to the servo shield. Each channel delivered 5V to each component. A schematic of the electrical setup is shown on page 8.

In addition, the initial design of incorporating batteries was scratched, as we were not confident if our battery setup would blow out the system. We took NiMH batteries and soldered several together, but this setup was not

rigorously tested out of concern for personal safety, and we decided to just use the much more reliable DC power supply.

Safety

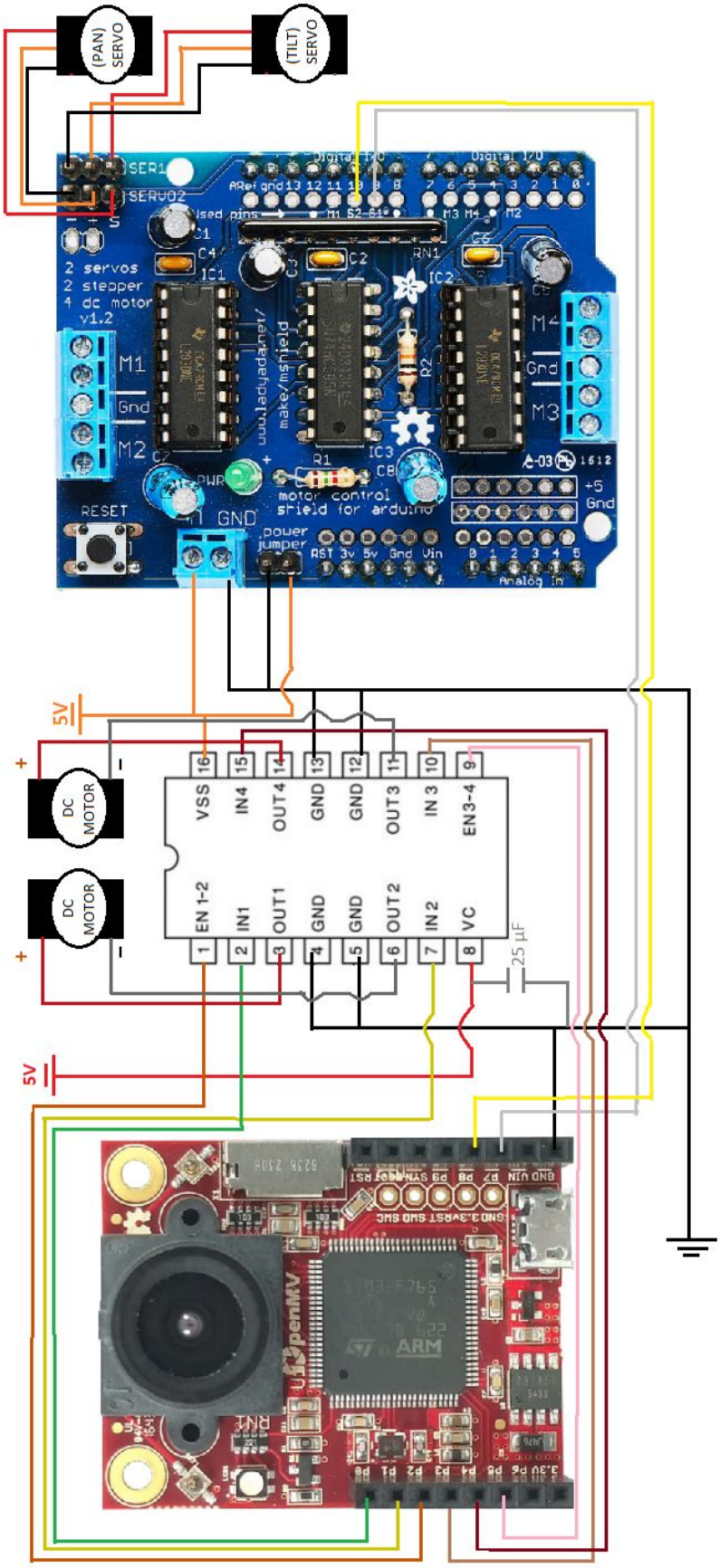
There is no major concern for the safety of the project. Because the robot's function is to follow an object, the only real risk is something getting in the way of the robot during operation.

As for the safety for each component, the voltage applied to each component will be within the manufacturer's design range. The voltage is supplied through batteries that run in series and will not exceed greater than 10V for a particular set. The circuits are grounded to common ground.

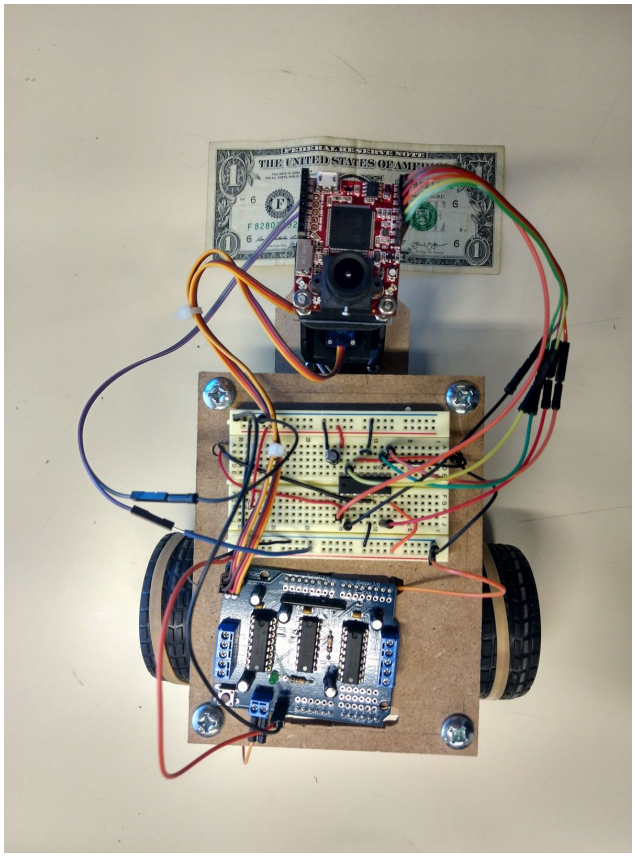
Acknowledgements

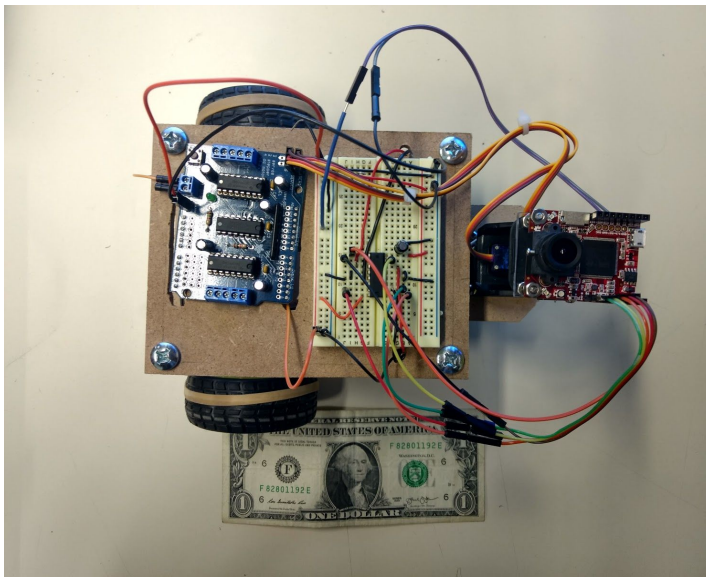
We would like to personally thank Professor Barriero and the TAs Rudy and Darius for immense help and guidance in this project and in the course as well. There were many times where we were stuck in our design and the guidance provided helped us move forward and ultimately allowed us to deliver a successful demonstration to the class. In addition, we would like to specifically thank Professor Barriero for providing us the OpenMV and the motors needed for the robot's construction. We'd like to specifically thank the TAs for teaching us how to use the machinery in the machine shop as our experience there was one of the most enjoyable parts of designing this project. We would not have been successful designing this project without each of their help.

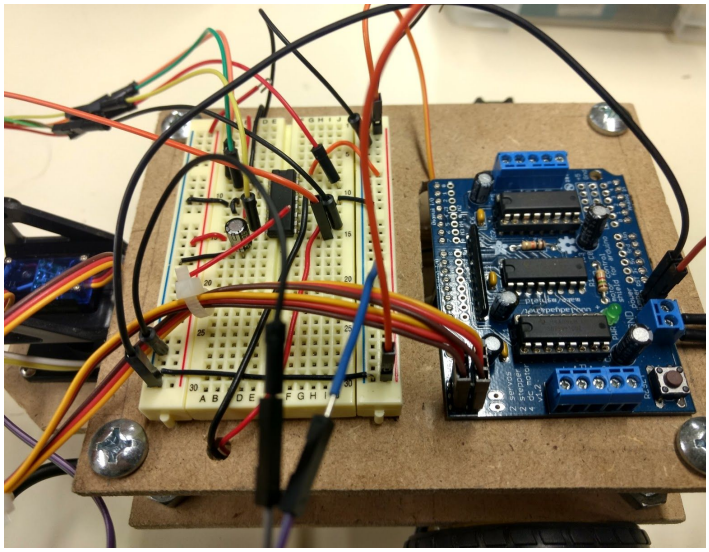
Electrical Diagram

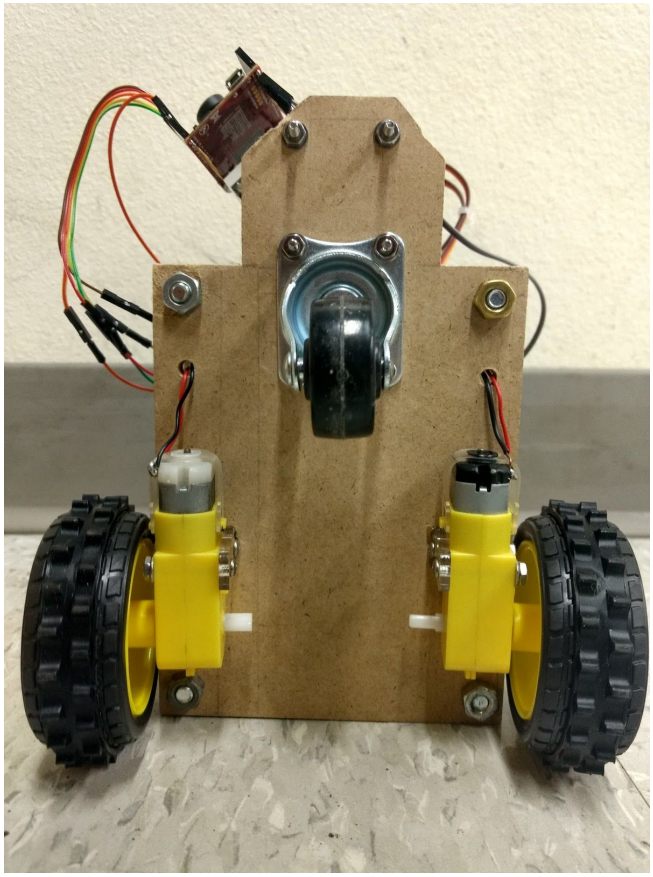


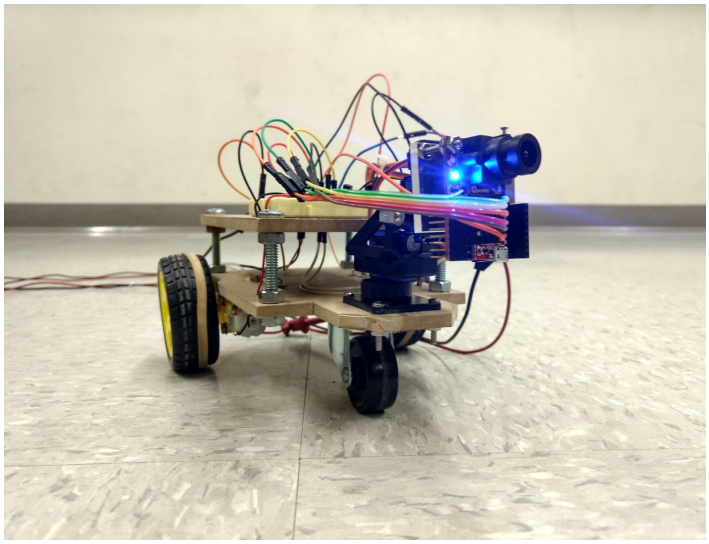
Complete Mechanical Assembly











Code Script

```
# MACHINE VISION ROBOT (MVR)
# AUTHORS: TREVOR McCLEEREY, JANSEN QUIROS
# PHYS 124 - FALL 2017
# DESCRIPTION: This program controls a robot that uses an OpenMV camera to track a green object.
#              In the demonstration,  we have chosen a green ball to be its focus. Two servo motors
#              are used for pan and tilt adjustment to center the camera's focus onto the ball. If
#              the ball cannot be focused because it is out of the camera's range, whether it is too
#              far to the right/left or too close/far, it will signal the DC motors to turn or move
#              forward or backwards.


import sensor, image, time
from pyb import Servo   # to initialize and control servos
from pyb import delay   # to use timing delay separate from time module
from pyb import Pin     # to set pin control
from pyb import LED

red_led   = LED(1)
green_led = LED(2)
blue_led  = LED(3)
ir_led    = LED(4)

# Color Tracking Thresholds (L Min, L Max, A Min, A Max, B Min, B Max)
# The below thresholds track in the green ball in different settings.
# You may pass up to 16 thresholds below. However, it's not really possible to segment any
# Scene with 16 thresholds before color thresholds start to overlap heavily.
thresholds = [(75, 93, -44, -24, -3, 7),   # Green facing desks
              (54, 88, -52, -26, 0, 20),   # Green facing whitboard
              (71, 91, -51, -29, -6, 8)]   # Green facing wall


sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
sensor.skip_frames(time = 2000)
sensor.set_auto_gain(False)     # Must be turned off for color tracking
sensor.set_auto_whitebal(False) # Must be turned off for color tracking
clock = time.clock()

# Only blobs that with more pixels than "pixel_threshold" and more area than "area_threshold" are
# returned by "find_blobs" below. Change "pixels_threshold" and "area_threshold" if you change the
# camera resolution. Don't set "merge=True" becuase that will merge blobs which we don't want here.
```

```
#########################################################################
# Servo initialization and calibration here

pan = Servo(1)  # use PWM on P7, s2 on shield
pan.calibration(540, 2400, 540, 1400, 1500)
pan.angle(90)

tilt = Servo(2) # use PWM on P8, s1 on shield
tilt.calibration(540, 2400, 1200, 1800, 1500)
tilt.angle(90)

delay(1000)     # delay to give servos time to get to 90 & 90
#########################################################################

#########################################################################
# DC Motor pin setup

LEFT_MOTOR1 = Pin('P0', Pin.OUT_PP, Pin.PULL_NONE)
LEFT_MOTOR2 = Pin('P1', Pin.OUT_PP, Pin.PULL_NONE)
LEFT_ENABLE = Pin('P2', Pin.OUT_PP, Pin.PULL_NONE) # HI -> ON;  LOW -> OFF

RIGHT_MOTOR1 = Pin('P3', Pin.OUT_PP, Pin.PULL_NONE)
RIGHT_MOTOR2 = Pin('P4', Pin.OUT_PP, Pin.PULL_NONE)
RIGHT_ENABLE = Pin('P5', Pin.OUT_PP, Pin.PULL_NONE) # HI -> ON; LOW -> OFF
#########################################################################

#########################################################################
# Constants initialization

# Blob sensing functions and constants begin here
# Area related constants and values:
LOW_AREA      = 8000  # likely needs adjustment
HIGH_AREA     = 11000 # likely needs adjustment
MIN_DELTA_A   = 200   # minimum change in area, knock out noise
last_area     = 0     # initialize to zero to start, updates with new data
new_area      = 0     # update in loop by taking measurement of blob

# Translational related constants and values
CENTER_CX           = 150
CENTER_CY           = 115
CENTERED_CX_THRESHOLD = 20
CENTERED_CY_THRESHOLD = 20
```

```
last_cx = 0
new_cx  = 0
last_cy = 0
new_cy  = 0

# Servo motor control constants
LOW_ANGLE    = 60
HIGH_ANGLE   = 120
PARALLEL_LOW  = 75
PARALLEL_HIGH = 105
pan_angle    = 90
tilt_angle   = 90
#########################################################################


#########################################################################
# This function checks if the pan servo is in the range near 90 degrees. This must be checked in
# order to move forward/reverse accurately. We only want forward motion if the MVR motion is parallel
# to the motion of the object
# PARAMETER - none
# RETURN - True: when the pan_angle is in the specified range around 90
#          False: when it's not in that range
# SIDE EFFECTS - none
def is_parallel():
  if pan_angle in range( PARALLEL_LOW, PARALLEL_HIGH ):
     blue_led.on()
     return True
  else:
     blue_led.off()
     return False


#########################################################################
# Area sensing function intended to detect a blob's change in area
# A change in area, delta_a, will be positive if the blob is getting further from the camera, and
# negative if the blob is getting closer to the camera. The former will require the DC motors
# to drive forward, the latter in reverse. This will maintain the ball in an acceptable area/distance
# PARAMETERS   - the_blob - a blob object used to measure the blob's area
# RETURN       - returns to caller only if delta_a is negligible; effectively
#                doing nothing. This is in order to eliminate noise in regard to the area
# SIDE EFFECTS - calls functions that drive the DC motors.
def sense_area( the_blob ):
```

```
  global diff_area, last_area     # Global allows you to modify these variables in this scope
  new_area  = the_blob.area()     # Get the blob's new area
  delta_a   = last_area - new_area   # Take difference
  last_area = new_area           # Set last area after difference has been taken
  print("area %d" % new_area)

  if is_parallel() and new_area not in range(LOW_AREA, HIGH_AREA):
     if new_area > HIGH_AREA:
        print("       REVERSE")
        reverse()
     elif new_area < LOW_AREA:
        print("FORWARD")
        forward()
  elif is_parallel() and new_area in range(LOW_AREA, HIGH_AREA):
        stop()


########################################################################
# This function uses the central cx position of 150 (approximately) and takes a measurement of the
# current cx, will update servos depending on how big the difference is from center
#
# PARAMETER - the_blob - a blob object
# RETURN - nothing when the blob object is not far off center
# SIDE EFFECTS - calls pan_adjust when the blob is far off center, in order to bring it back to center
def center_cx( the_blob ):
  off_center = CENTER_CX - the_blob.cx()
  if abs( off_center ) < CENTERED_CX_THRESHOLD:
     return # Close enough to center
  else:
     check_pan_extreme()
     pan_adjust( off_center )


########################################################################
# This function uses the central cy position of 150 and takes a measurement of the current cy. Then
# it will send the amount that the blob is off center to the servo tilt control to bring it back to
# center.
#
# PARAMETER: the_blob - a blob object
# RETURN - nothing when the blob is near the center of image
# SIDE EFFECTS - calls tiltAdjust when the blob is vertically off center
def center_cy( the_blob ):
  off_center = CENTER_CY - the_blob.cy()  # error value
```

```
  if abs( off_center ) < CENTERED_CY_THRESHOLD:
    return # close enough to center
  else:
    check_tilt_extreme()
    tiltAdjust( off_center )


#####################################################################
# this function will check if the pan servo is getting near its maximum range, either high or low
# if the pan is approaching the low extreme ( zero degrees on pan_angle ) then the vehicle needs to
# pivot right. This will aid the servos in keeping a blob centered in the image without reaching
# the max and not being able to turn any more
def check_pan_extreme():
  # Check low extreme
  if pan_angle < LOW_ANGLE:
    green_led.on()
    pivot_right()
  # Check high extreme
  elif pan_angle > HIGH_ANGLE:
    red_led.on()
    pivot_left()
  else:
    green_led.off()
    red_led.off()
    stop() # The pan is not near the extremes


#####################################################################
# The main thing to consider for tilt being at an extreme is probably going to be for an object
# directly under ( or nearly under ) the front of the vehicle. When this is the case, we can bring
# the object into better view by rolling directly backward away from the object. Probably no need
# to bother with the object being near the high extreme, as the vehicle's purpose is to track objects
# on the ground
def check_tilt_extreme():
  # Only check low extreme
  if tilt_angle not in range(LOW_ANGLE, HIGH_ANGLE):
    reverse()
  else:
    stop()

#####################################################################
# This function adjusts the pan servo which will correspond to changes in a blob's x position
# It also ensures that the angle being set does not exceed the maximum range of the servo
```

```
# PARAMETER - deltaX - some value that indicates how much the blob has changed its X pos.
# RETURN - none
# SIDE EFFECTS - updates the value of pan_angle and also adjusts the value on pan.angle()
def pan_adjust( deltaX ):
  global pan_angle       # need 'global' to modifiy in this scope
  pan_angle += ( deltaX // 12 )
  # maintain the value of pan_angle between 0 and 180.
  if pan_angle < 0:
     pan_angle = 0
  elif pan_angle > 179:
     pan_angle = 179
  pan.angle(pan_angle) # write the new angle


#########################################################################
# This function adjusts the tilt servo which will correspond to changes in a blob's y position
# It also ensures that the angle being set does not exceed the maximum range of the servo
# PARAMETER - deltaY - some value that indicates how much the blob has changed its Y pos.
# RETURN - none
# SIDE EFFECTS - updates the value of tilt_angle and also adjusts the value on tilt.angle()
def tiltAdjust( deltaY ):
  global tilt_angle     # need 'global' to modifiy in this scope
  tilt_angle -= (deltaY // 12)
     # These checks keep our angle within the allowable range, so it doesn't get messed at the
     # limits. Basically do not allow angles less than 0 or greater than 180
  if tilt_angle < 0:
     tilt_angle = 0
  elif tilt_angle > 149:
     tilt_angle = 149
  tilt.angle(tilt_angle)


#########################################################################
# This function sets both of the enable pins to low, which will stop both motors from doing
# whatever they are doing
# PARAMETER - none
# RETURN - none
# SIDE EFFECTS - sets pin2/pin5 to low state
def stop():
  RIGHT_ENABLE.low()
  LEFT_ENABLE.low()
```

```
#########################################################################
# This function will pivot the vehicle clockwise, or right from the perspective of the vehicle
# to achieve a pivot motion, the right DC motor will be rolled backward, and the left forward
def pivot_right():
  left_motor( -1 )
  right_motor( 1 )


#########################################################################
# This function will pivot the vehicle counter clockwise, or left from the perspective of the vehicle
# to achieve a pivot motion, the left DC motor will be rolled backward, and the right forward
#This function will pivot the
def pivot_left():
  left_motor( 1 )
  right_motor( -1 )


#########################################################################
# This function will turn both DC motors in reverse at the same rate, in order to move the vehicle
# backward.
def reverse():
  left_motor( -1 )
  right_motor( -1 )

def forward():
  left_motor( 1 )
  right_motor( 1 )

def left_motor( direction ):
  if direction < 0: # turn wheel forward
     LEFT_MOTOR1.high()
     LEFT_MOTOR2.low()
  else: # turn wheel backward
     LEFT_MOTOR1.low()
     LEFT_MOTOR2.high()
  LEFT_ENABLE.high()

def right_motor( direction ):
  if direction < 0:
     RIGHT_MOTOR1.low()
     RIGHT_MOTOR2.high()
  else:
     RIGHT_MOTOR1.high()
```

```
    RIGHT_MOTOR2.low()
  RIGHT_ENABLE.high()

while(True):
  clock.tick()
  img = sensor.snapshot()

  blob_list = img.find_blobs(thresholds, pixels_threshold=100, area_threshold=100, merged=True)
  if len(blob_list) == 0:
    # Only want 1 blob in our image. If more than 1 continue to next loop
    stop()
    continue

  # Set big blob to first element
  bigBlob = blob_list[0]
  # then loop over all blobs to find the largest in the list
  for blob in blob_list:
    if bigBlob.area() < blob.area():
      bigBlob = blob

  center_cx( bigBlob ) # Keep x coord of blob near 150 degrees
  center_cy( bigBlob ) # Keep y coord of blob near 150 degrees
  sense_area( bigBlob ) # Monitor the distance of the blob
  print( pan_angle )
  # Draw box with cross on each blob
  img.draw_rectangle(bigBlob.rect())
  img.draw_cross(bigBlob.cx(), bigBlob.cy())
```