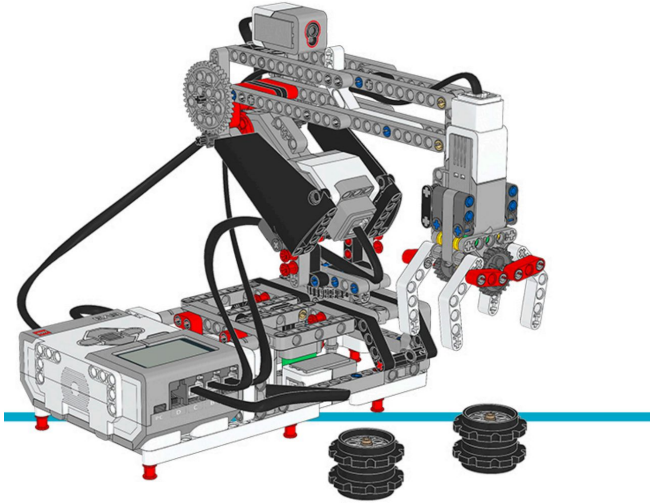
Object Sorter

Leah Ersoy, Stuart Harley, Nathan Chapman

Introduction

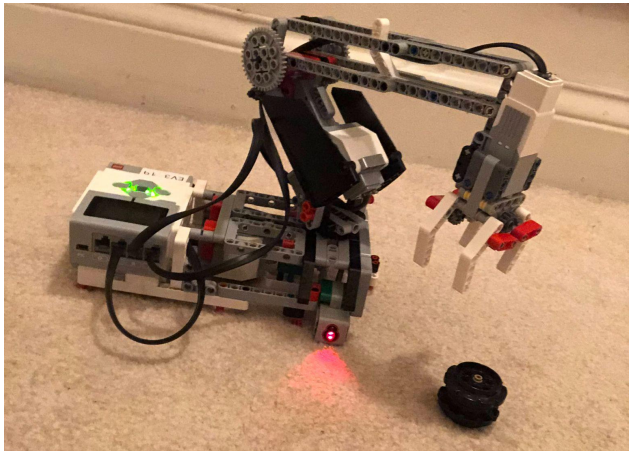
- Robot arm built via EV3 blueprints with a modified color sensor location
- Sorts objects by color
- Uses computer vision to locate objects



OpenCV

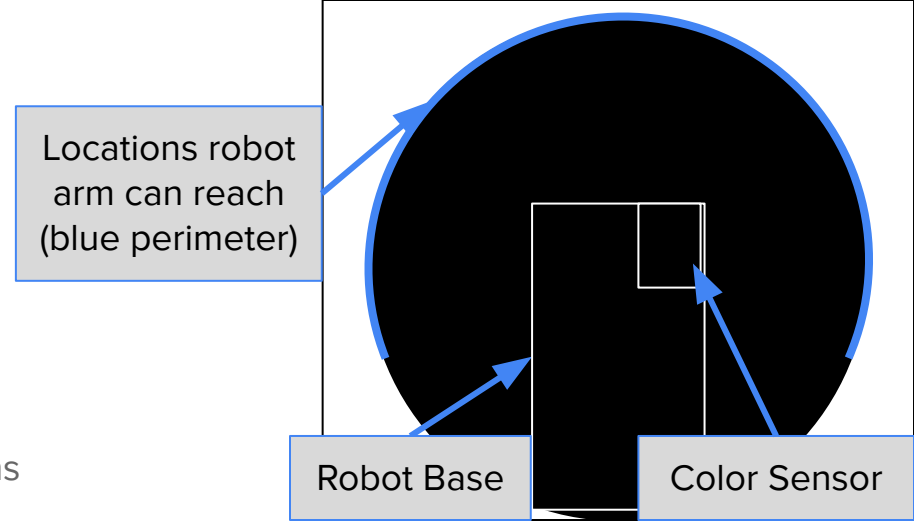
Real world

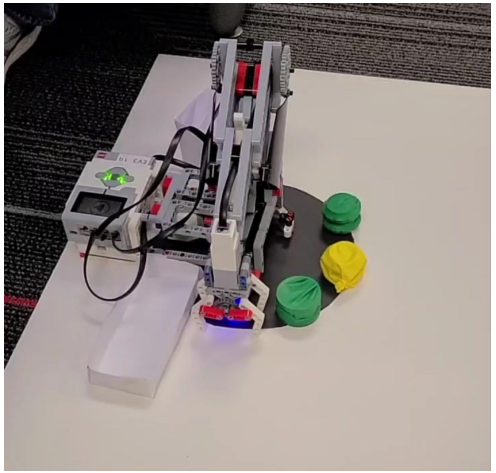
- Similar to robotic arms used in real-world assembly lines
- Sort objects based on size, color, shape, etc
- Limitations in our components cause our arm to be less functional than those used in industry



Background

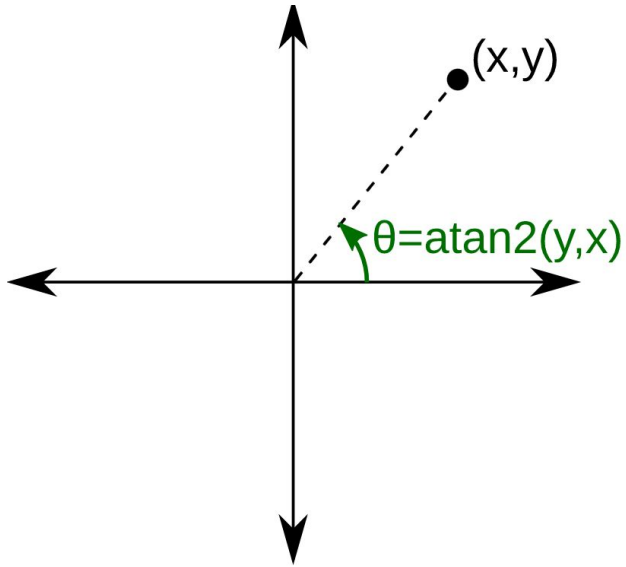
- Server/Client relationship
  - Robot as server
- Camera watches playing field
  - Process using OpenCV to get object locations
  - Send packets containing locations to robot
- Arm picks up objects and reads color
  - If color has a bin moves it there, or places in default bin





Background Continued

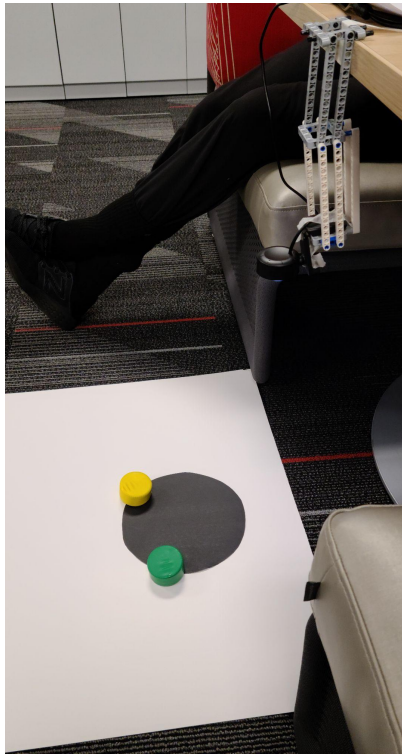
- Arm has one degree of freedom
  - Can rotate a little over 180 degrees
- Arm centered on the origin
- atan2 rather than atan
  - atan gives angle value between -90 to 90
  - atan2 gives angle value between -180 to 180
    - atan2(y - origin_y, x - origin_x)



Results Step 1: Computer Vision

- Fixed overhead camera
- Have to begin by calibrating the camera
  - Find centroid of the playing field
  - Find the centroids of the objects
- Determine if the objects are within reach of the arm
- Find the angles to the objects

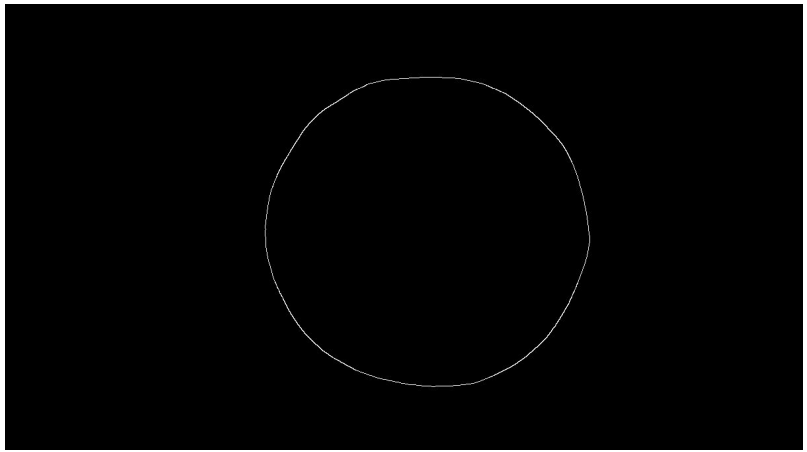
```
Centroid:  660 364
Object Locations:  [(705, 133), (530, 150), (869, 190)]
Object Angles:  [78.97654403625687, 121.27770286686645, 39.77859568017496]
```



Find the Locations the arm can reach

- Use cv.GaussianBlur() and cv.Canny()

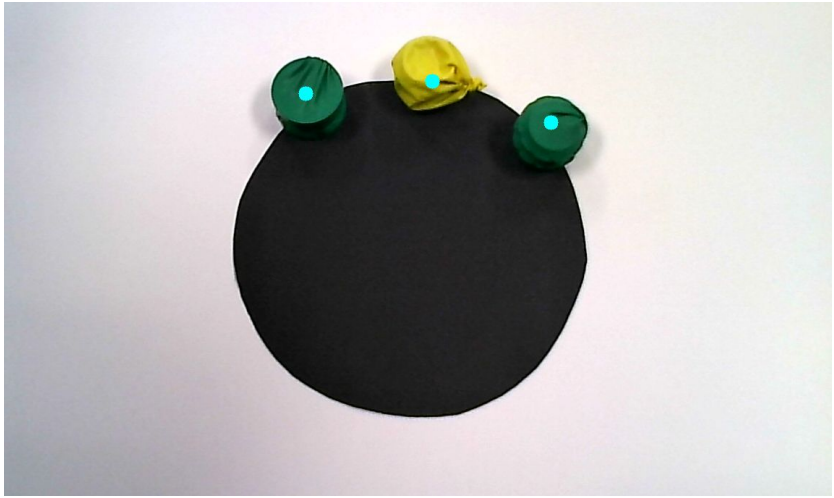
```
img_blur = cv.GaussianBlur(resize, (5, 5), cv.BORDER_DEFAULT)
edges = cv.Canny(image=img_blur, threshold1=100, threshold2=200)
```



Locating Objects

- Use thresholding to get rid of anything white or black
- Find centroids of what's left
  - cv2.connectedcomponentswithstats()
  - Ensure components are within size range
  - Ensure components are within range of arm

```
for i in range(0, numLabels):
    isIn = False
    x, y = centroids[i]
    x = int(x)
    y = int(y)
    #print(stats[i, cv.CC_STAT_AREA])
    #If it is around the area of a tire
    if stats[i, cv.CC_STAT_AREA] > 3000 and stats[i, cv.CC_STAT_AREA] < 15000:
        for i in range(-25,25):
            for n in range(-25,25):
                if edges[y+i,x+n] == 255:
                    isIn = True
        if isIn:
            cv.circle(resize, (int(x), int(y)), 10, (255, 255, 0), -1)
            objectLocations.append((x,y))
```

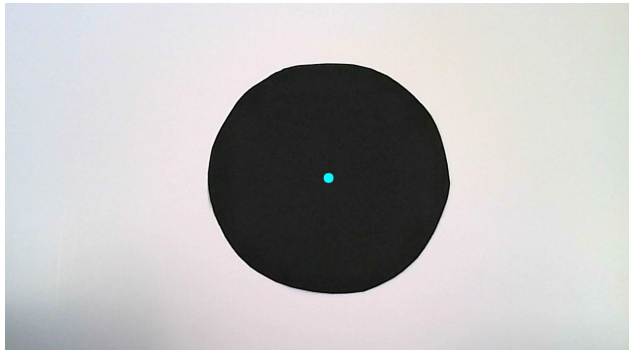


Determine Angles to the Objects

- Use cv.inRange() to find the centroid of the black circle
- The centroid of the playing field and centroids of the objects are used to find the angle

```
frame_HSV = cv.cvtColor(resize, cv.COLOR_BGR2RGB)
frame_threshold = cv.inRange(frame_HSV, (0,0,0), (40,40,40))
```

```
#Finds all the angles to the objects on the playing field
def findAngles():
    for point in objectLocations:
        x,y = point
        theta = math.degrees(math.atan2(y - cY, x - cX))
        theta = abs(theta)
        objectAngles.append(theta)
```



Results Step 2: Arm Functionality

- Defined Simple Server/Client movement

```
if cmd == "raise":
    raise_arm_motor.run_target(DEG_PER_SEC, -270)
    mbox.send('ready')
elif cmd == "lower":
    raise_arm_motor.run_target(DEG_PER_SEC, 0)
    mbox.send('ready')
elif cmd == "close":
    claw_motor.run_target(DEG_PER_SEC, 90)
    mbox.send('ready')
elif cmd == "open":
    claw_motor.run_target(DEG_PER_SEC, 0)
    mbox.send('ready')
elif cmd == "grab":
    claw_motor.run_until_stalled(DEG_PER_SEC)
    claw_motor.run_time(DEG_PER_SEC, .3)  # Grips object tighter
    mbox.send('ready')
elif cmd == "rotate":
    # 0 degrees is directly infront of color sensor
    # gear ratio is 12/36
    angle = float(arg)
    spin_arm_motor.run_target(-DEG_PER_SEC, -angle*3)
    mbox.send('ready')
elif cmd == "color":
    mbox.send(color_sensor.color())
else:
    print("Received invalid cmd "+cmd)
```

```
def client():
    client = BluetoothMailboxClient()
    mbox = TextMailbox('greeting', client)

    print('establishing connection...')
    client.connect(SERVER)
    print('connected!')
    while 1:
        cmd = input("Enter a cmd (raise, lower, close, open, grab, rotate, color): ")
        if cmd == 'raise':
            mbox.send(cmd + ':')
        elif cmd == 'lower':
            mbox.send(cmd + ':')
        elif cmd == 'close':
            mbox.send(cmd + ':')
        elif cmd == 'open':
            mbox.send(cmd + ':')
        elif cmd == 'grab':
            mbox.send(cmd + ':')
        elif cmd == 'rotate':
            degrees = input("Rotate to what degree angle: ")
            msg = cmd + ':' + degrees
            mbox.send(msg)
        elif cmd == 'color':
            mbox.send(cmd + ':')
            mbox.wait()
            color = mbox.read()
            print("Color detected: ", color)
        else:
            print("Unrecognized cmd: "+cmd)
```

Results Step 2: Arm Functionality

- Created a method to sort colored objects
- Pass in the number of objects, tuple of angles where objects are located, and a color to sort on
- For our tests, we sorted out the color green
- Limited by our range of movement the number of colors we could sort by

```
def sort(num_objects, angles, sort_color):
    client = BluetoothMailboxClient()
    mbox = TextMailbox('greeting', client)

    print('establishing connection...')
    client.connect(SERVER)
    print('connected!')
    mbox.send('raise:')
    mbox.wait()
    for i in range(num_objects):
        mbox.send('rotate:' + str(angles[i]))
        mbox.wait()
        mbox.send('lower:')
        mbox.wait()
        mbox.send('grab:')
        mbox.wait()
        mbox.send('raise:')
        mbox.wait()
        mbox.send('rotate:0')
        mbox.wait()
        mbox.send('lower:')
        mbox.wait()
        mbox.send('color:')
        mbox.wait()
        color = mbox.read()
        print('Color detected: ', color)
        mbox.send('raise:')
        mbox.wait()
        if color == 'Color.' + sort_color:
            mbox.send('rotate:190')
        else:
            mbox.send('rotate:-30')
        mbox.wait()
        mbox.send('open:')
        mbox.wait()
    mbox.send('rotate:0')
    mbox.wait()
    mbox.send('lower:')
    mbox.wait()
```

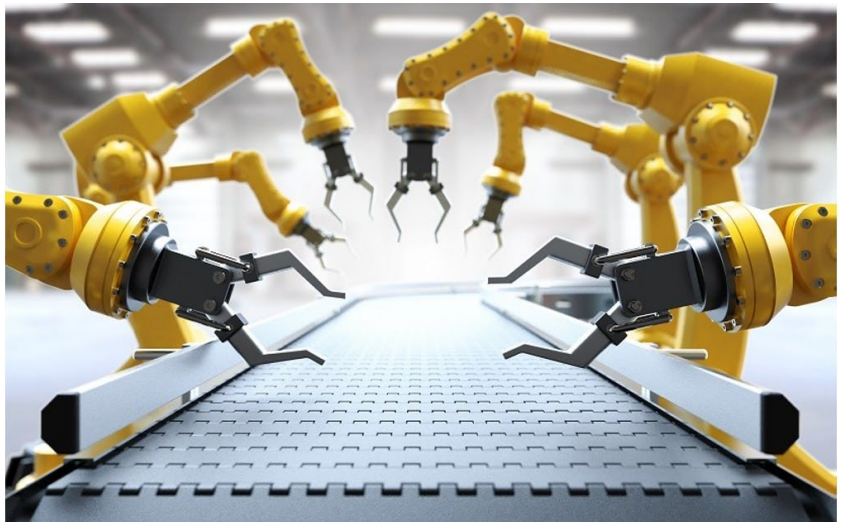
Demonstration

Limitations

- Computer Vision does not handle noise well - Lighting/Shadows, Irregularities in surface color
- Objects must be a specific size for claw to pick up
- Cables were not long enough to mount the color sensor to the claw
- Color sensor is inconsistent if object is not positioned centrally in claw
- Only have 2 motors - Arm only has 1 degree rotational motion since it also moves up/down.
- Arm cannot rotate 360 degrees because base and wires become obstacles
- The base is not mounted so it can move and is not perfectly centered
- We only do Computer Vision when the arm is not present because it would create too much noise if present

Discussion/Conclusion

- A more robust system with better components would be able to handle noise better and move in all directions.
- Robotic arms similar to these are used in industry.



References

https://www.pyimagesearch.com/2021/05/12/opencv-edge-detection-cv2-canny/

https://docs.opencv.org/4.5.4/

https://le-www-live-s.legocdn.com/sc/media/lessons/mindstorms-ev3/building-instructions/ev3-model-core-set-robot-arm-h25-56cdb22c1e3a02f1770bda72862ce2bd.pdf

https://en.wikipedia.org/wiki/Atan2