Team Lucid

>>Renegade

The Team

Justin McGettigan	**David Ciccarello**	**Alex Olinger**
Team Lead	Hardware Integration	Testing
Lead Developer	Programmer	Debugging
	Mathematician	

The Challenge

Parking	Park car in front of a cone.
Collision Avoidance	Move to the left or right of a cone instead of into it.
Serpentine	Swivel through a series of cones.
Polebending	Drive parallel, perform U-turns, and execute serpentine.

Parking

- Detect an object by contourArea.

- Determine center of object.

- If center of object goes below the middle of the image, stop.

- Use the pixel offset from the middle of the image to determine the error/steering angle (such that the car steers towards the cone).

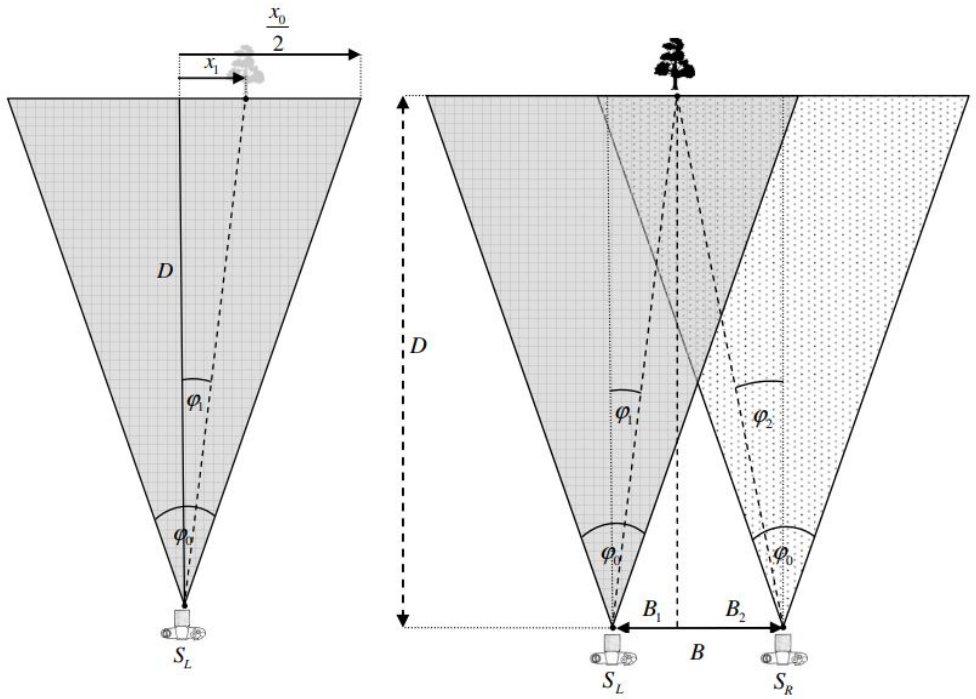
```
def control(self, left, right):
    stop = False
    height, width = left.getImage().shape[:2]
    lCenter, rCenter = left.getCenter(), right.getCenter()
    lBounds, rBounds = left.getBounds(), right.getBounds()
    center = (lCenter[0] + rCenter[0]) / 2, (lCenter[1] + rCenter[1]) / 2
    bottomBound = (lBounds[1] + rBounds[1]) / 2
    if lCenter[0] == 0: center = rCenter
    if rCenter[0] == 0: center = lCenter
    coneSeen = center[0] > 0

    if center[1] > height/4:
        stop = True
    error = -(center[0] - (width/2))*(2/336.0)
    print error
    steeringAngle = error
    if steeringAngle > 0.3: steeringAngle = 0.3
    if steeringAngle < -0.3: steeringAngle = -0.3
    speed = 1
    return self.decide(speed, steeringAngle, stop)
```

Parking

$$\frac{\frac{x_1}{x_0}}{2} = \frac{\tan\varphi_1}{\tan\left(\frac{\varphi_0}{2}\right)},$$

$$D = \frac{B}{\tan\varphi_1 + \tan\varphi_2}$$

$$D = \frac{Bx_0}{2\tan\left(\frac{\varphi_0}{2}\right)(x_L - x_D)}$$



http://dsc.ijs.si/files/papers/s101%20mrovlje.pdf

Object Avoidance

- The same as parking except instead of stopping you simply switch the steering so that you go away from the cone instead of towards it.

```
def control(self, left, right):
    stop = False
    height, width = left.getImage().shape[:2]
    lCenter, rCenter = left.getCenter(), right.getCenter()
    center = (lCenter[0] + rCenter[0]) / 2, (lCenter[1] + rCenter[1]) / 2
    if lCenter[0] == 0: center = rCenter
    if rCenter[0] == 0: center = lCenter
    coneSeen = center[0] > 0

    if center[1] > height/4:
        error = (center[0] - (width/2))*(2/336.0)
    error = -(center[0] - (width/2))*(2/336.0)
    steeringAngle = error
    if steeringAngle > 0.3: steeringAngle = 0.3
    if steeringAngle < -0.3: steeringAngle = -0.3
    speed = 1
    return self.decide(speed, steeringAngle, stop)
```

Serpentine

- Look at where the largest contourArea (blob of color) on the screen is.

- Do this for both the cubes and cones simultaneously (by color).

- Determine whether to use the cube or the cone as the object of focus based on which is closer.

- Whichever side the object is on, is the side we focus on (with the lidar) while ignoring the other side.

- Aim for a desired distance from the object that is perpendicular with the car (perpendicular mode).

- Once the object reaches a certain angle to the car, the car goes around the cone (orbit mode).

- Once the next cone moves from the right side of the image to the right side of the image, re-enter perpendicular mode.

Polebending

- Use perpendicular mode constantly until a cube is seen. Count the cones while doing this.

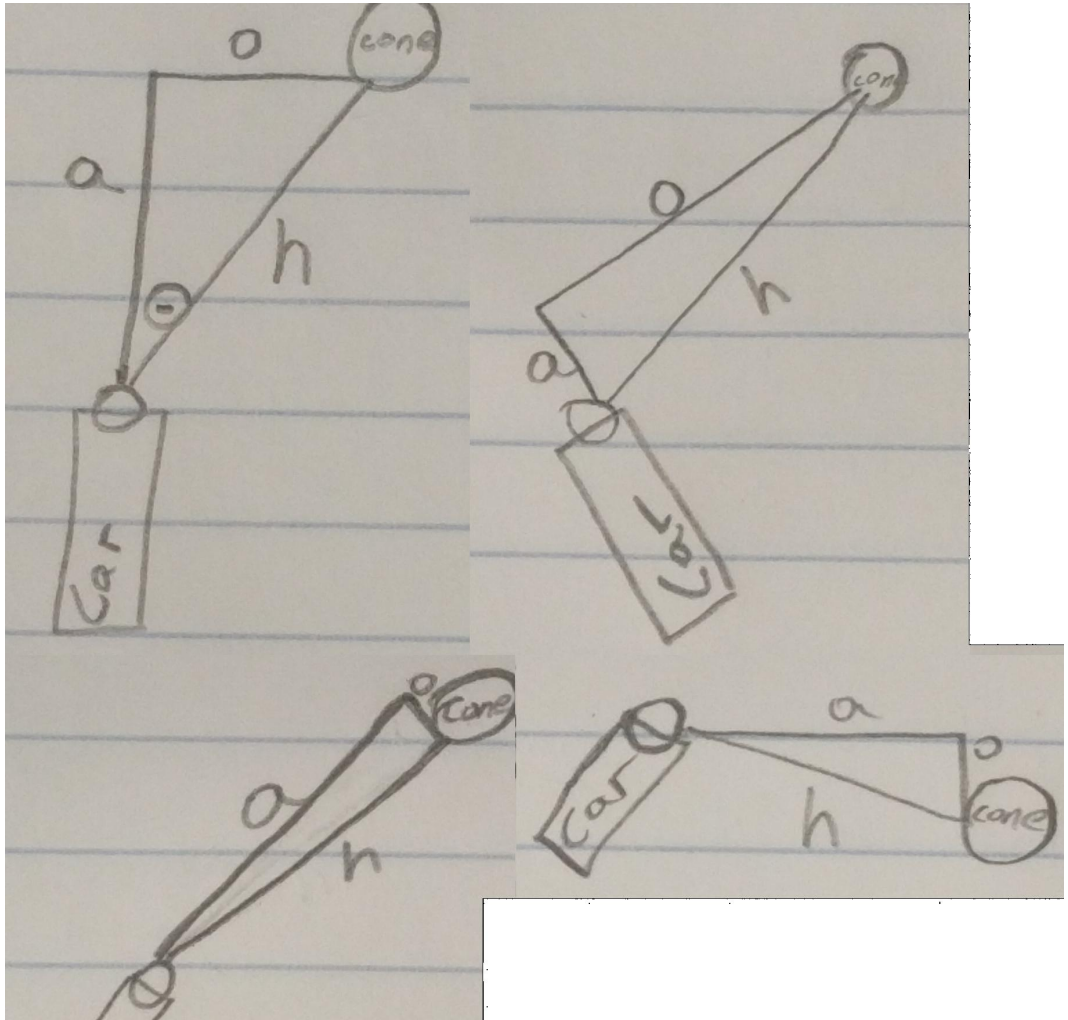
- Orbit around the cube.

- Enter serpentine mode and go until you reach the number of cones you counted.

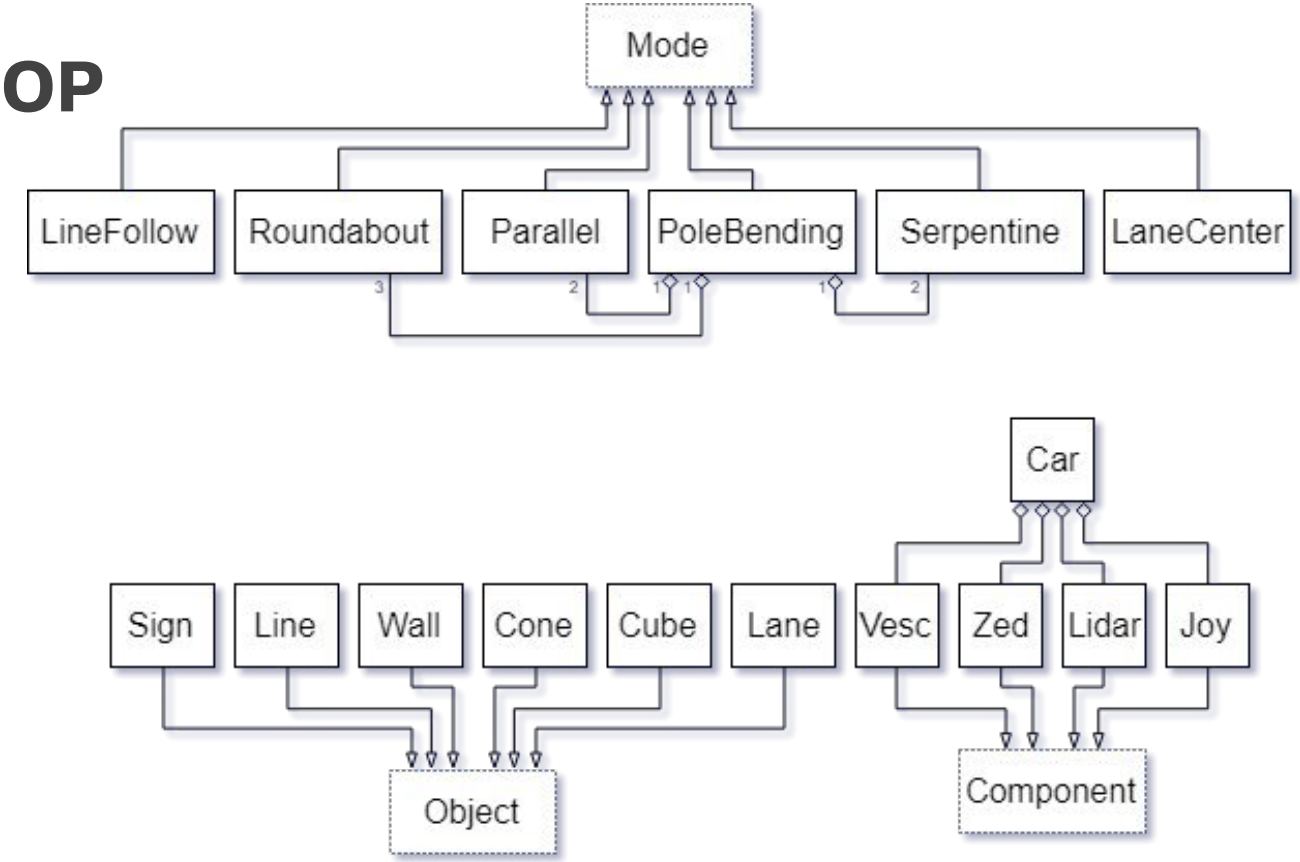
- Orbit around the last cone and continue serpentine mode until a cube is seen.

- Orbit around the cube.

- Go back to constantly suing perpendicular mode and stop when no more cones are seen.


o
cone
a
h
car
o
cone
h
a
car
o
cone
a
h
car
a
o
h
cone

OOP


Mode
LineFollow
Roundabout
Parallel
PoleBending
Serpentine
LaneCenter
3
2
1
1
1
2
Car
Sign
Line
Wall
Cone
Cube
Lane
Vesc
Zed
Lidar
Joy
Object
Component

Lessons Learned

- Try not to overthink your problems.

The End

```
def __init__(self):
    """Initialize the components of the car."""
    rp.init_node("car", anonymous=True)
    self.joy = Joy()
    self.zed = Zed()
    self.lidar = Lidar()
    self.vesc = Vesc()

    rp.Subscriber("vesc/joy", JoyMsg, self.joy_callback)
    rp.Subscriber("zed/normal", Image, self.zed_callback)
    rp.Subscriber("scan", LaserScan, self.lidar_callback)
def joy_callback(self, data):
    self.joy.setData(data)


def zed_callback(self, data):
    self.zed.setImage(data)
    self.controller(self.zed, self.lidar, self.vesc)


def lidar_callback(self, data):
    self.lidar.setData(data)
```