README:

All the vectors and their operations are included in the class “Vector” under src/utilities/Vector. These operations are dot product, magnitude, add, subtract, getAngle, unitVector, multiply, and divide.

All three tasks for this project require controllers to ultimately modify the control variables of the car object which are going to be used in the car class update method. All three tasks would affect the acceleration in their own way, but the modifying the control variables from there remains the same, so we can use one method for modifying control variables in all three tasks, this method is made under the class OutputFiltering(src/utilities/OutputFiltering) as moveCar(). This method takes in three variables, Car car, Vector acceleration, and double[][] control variables.

In this method, we would first find the projection of A, by using a dot product between the car position vector and acceleration vector. If the projection of A is greater than zero, then set the throttle to 1. If the projection of A is less than -1, then set the brake to 1. This -1 acts as a threshold in cases where the negative is too close to zero. Also, the right perpendicular vector is calculated using the formula of the rotating vector given below. This right vector is used in the calculation of variable steering by getting the dot product value of this vector with the acceleration vector. Whatever the value of this dot product, positive or negative not zero, that value is divided by 3 and set as variable steering. This division by 3 acts as a threshold for variable steering so that small values won’t affect the variables steering much.

// Formula of rotating a vector

// x2 = cosB x1 - sinB y1 (i.e for Pi/2; x2 = -y1)

// y2 = sinB x1 + cosB y1 (i.e for Pi/2; y2 = x1)

Seek Controller:

Now, general stuff out of our way. Let’s discuss the Seek Controller. Seek Controller class exists under the controllers directory and extends Controller class. The code used in the update method of this controller to find out the acceleration has been given in the lecture slides. Once calculated acceleration vector A, control variables are calculated by calling OutputFiltering class and passing acceleration vector A, Car reference, and control variables array reference to moveCar() method. Thus, seek controller works with this above description.

Arrive Controller:

Now, let’s discuss the Arrive Controller, Arrive Controller class exists under the controllers directory and extends the Controller class. This class is initialized with the arguments target reference, target radius, and slow radius. The code used in the update method of this controller to find out the acceleration vector A based on the position of the car with respect to both the radii. These radii affect the target speed from which acceleration vector A is calculated. Once

calculated acceleration vector A, control variables are calculated by calling OutputFiltering class and passing acceleration vector A, Car reference, and control variables array reference to moveCar() method. Thus, arrive controller works with this above description.

Wall avoidance Controller:

Now, let's discuss the Wall Avoidance Controller which exists under the controllers directory and extends the Controller class. To implement this, we need to have ray casting for the left, front, and right directions. This ray casting is implemented using a class under Output Filtering as RayCast. This class initializes by taking three arguments, Buffered image m_img used to fake transparent box, int measure length used in as the length seeking in the direction, deflect angle used in setting the direction. This class has a cast() method which has a for loop for incrementing distance alongside the ray by 1 by moving that fake transport box alongside a ray in the deflect angle direction. This cast() method returns a boolean which tells the controller in which direction, there is collision detected upfront.

In the controller update method, three rays are initialized with zero degrees deflect angle as the front, -Pi/4 as left, and Pi/4 as a right ray. These rays are projected using the cast() method of the ray cast class. If any of the rays found a collision, we would seek towards the opposite direction, for example, if the left cast found a collision, seek to right cast deflected direction, or if the right cast found a collision, seek to left cast deflected direction. This deflection vector is calculated by the avoidCollision() method that exists in the ray cast class. With this director vector calculated, the acceleration vector is calculated by normalizing the director vector and multiplying it by the maxAcceleration. Once calculated acceleration vector A, control variables are calculated by calling OutputFiltering class and passing acceleration vector A, Car reference, and control variables array reference to moveCar() method. Thus, the wall avoidance controller works with the above description.