- Modify sumo_surpervise.py in Webots\ projects\ default\controllers\ sumo_supervisor to add new type vehicle: realworld_vehicle into webots, which is similar to webots_vehicle which already exis.(all code are in sumosupervisor.zip)

1. The places marked in red are the ones to add/modify, and searching for the code in black will lead you to the general location.

```
def get_initial_vehicles(self):
    """Get all the vehicles (both controlled by SUMO and Webots) already present in the world."""
    for i in range(0, self.vehiclesLimit):
        defName = "SUMO_VEHICLE%d" % self.vehicleNumber
        node = self.getFromDef(defName)
        if node:
            self.vehicles[i] = Vehicle(node)
            self.vehicles[i].name.setSFString("SUMO vehicle %i" % self.vehicleNumber)
            self.vehicleNumber += 1
        else:
            break
    for i in range(0, self.vehiclesLimit):
        print("webots = ", i)
        defName = "WEBOTS_VEHICLE%d" % self.webotsVehicleNumber
        node = self.getFromDef(defName)
        if node:
            self.webotsVehicles[i] = WebotsVehicle(node, self.webotsVehicleNumber)
            self.webotsVehicleNumber += 1
        else:
            break
    print(self.webotsVehicleNumber)

    for i in range(0, self.vehiclesLimit):
        # self.realworldNumber = 0
        print("real = ", i)
        defName = "REALWORLD_VEHICLE%d" % self.realworldNumber
        node = self.getFromDef(defName)
        if node:
            self.realworldVehicle[i] = RealworldVehicle(node, self.realworldNumber)
            self.realworldNumber += 1
        else:
            break
    print(self.realworldNumber)
```

```
        self.traci = traci
        self.sumolib = sumolib
        self.radius = radius
        self.enableHeight = enableHeight
        self.sumoClosed = False
        self.temporaryDirectory = directory
        self.rootChildren = self.getRoot().getField("children")
        self.viewpointPosition = self.get_viewpoint_position_field()
        self.maxWebotsVehicleDistanceToLane = 5
        self.webotsVehicleNumber = 0
        self.webotsVehicles = {}
        self.vehicleNumber = 0
        self.vehicles = {}
        self.vehiclesLimit = maxVehicles
        self.vehiclesClass = {}
        self.realworldNumber = 0
        self.realworldVehicle = {}
        self.maxRealworldVehicleDistanceToLane = 5
```

2. Map to the SUMO interface：**SumoSupervisor.py**

```
            self.update_vehicles_position_and_velocity(step, rotateWheels)
            self.update_webots_vehicles(xOffset, yOffset)
            self.update_realworld_vehicles(xOffset, yOffset)
```

3. **Add RealworldVehicle.py**（similar to WebotsVehicle.py）

```
import math
class RealworldVehicle:
    """Class that defines a vehicle controlled by Webots."""

    def __init__(self, node, id):
        self.previousPosition = [0, 0, 0]
        self.vehicleLength = 6
        self.vehicleHeight = 0.4
        self.node = node
        self.name = "realworldVehicle%d" % id
```

4. About "Double Shadow “ Problem Solving: sumoSupervisor.py （Because in the Webots world, the coordinates of the car are mapped to the SUMO interface in two dimensions. But SUMo itself will also generate SUMO type vehicles. When the SUMO is reflected but the Webots interface does not have the corresponding serial number, it will go back to the projection without approval, resulting in the overlap of two cars in the same position.）

```
# subscribe to new vehicle
            for id in result[self.traci.constants.VAR_DEPARTED_VEHICLES_IDS]:
                if not (id.startswith("webotsVehicle") or id.startswith("realworldVehicle")):
                    self.traci.vehicle.subscribe(id, self.vehicleVariableList)
                elif self.sumoDisplay is not None and len(self.webotsVehicles) == 1:
                    # Only one vehicle controlled by Webots => center the view on it
                    self.traci.gui.trackVehicle(view, 'webotsVehicle0')
```

- By creating a controller to control the RealWorld vehicle, in Webots\projects\vehicles \controllers\realworld, the GPS data in EXE can be dynamically read in real time, and the position and Angle can be updated synchronously in Webots.(all code are in realworld.zip)

1. Installing the EXE file: Unzip SocketServer v0.3.zip

2. Change the path to exe in socket_client.py, line 52. Connect to EXE and read the data

3. Move the position and change the angle： **realworld.py**

```
robot_node = supervisor.getFromDef("REALWORLD_VEHICLE0")
trans_field = robot_node.getField("translation")
rot_field = robot_node.getField("rotation")
optParser = optparse.OptionParser(usage="usage: %prog --input=file.osm [options]")

......
 trans_field.setSFVec3f(POSITION)
        rot_field.setSFRotation(rotation)
        robot_node.resetPhysics()
```

(a) Latitude and longitude and Cartesian coordinate formula conversion, unit conversion and Angle conversion formula

```
    lat0 =31.27375
    long0 = 120.7353

    utm_zone = 1 + math.floor((float(long0) + 180) / 6)
    hemisphere = 'south' if lat0 < 0 else 'north'
    projectionString = \
        "+proj=utm +%s +zone=%d +lon_0=%f +lat_0=%f +x_0=0 +y_0=0 +ellps=WGS84 +units=m +no_defs" % \
        (hemisphere, utm_zone, long0, lat0)

    utm_coord_zero = pyproj.Proj(projectionString)

xoffest, yoffset = utm_coord_zero(long0, lat0)
```

```
 yaw = float(s.data_rev['angel'])
        roll = 0
        pitch = 0

        a = math.cos(roll) * math.cos(yaw)
        b = -math.sin(roll)
        c = math.cos(roll) * math.sin(yaw)
        d = math.sin(roll) * math.cos(yaw) * math.cos(pitch) + math.sin(yaw) *
math.sin(pitch)
        e = math.cos(roll) * math.cos(pitch)
        f = math.sin(roll) * math.sin(yaw) * math.cos(pitch) - math.cos(yaw) *
math.sin(pitch)
        g = math.sin(roll) * math.cos(yaw) * math.sin(pitch) - math.sin(yaw) *
math.cos(pitch)
        h = math.cos(roll) * math.sin(pitch)
        i = math.sin(roll) * math.sin(yaw) * math.sin(pitch) + math.cos(yaw) *
math.cos(pitch)

        cosAngle = 0.5 * (a + e + i - 1.0)

        rotation[0] = 0
        rotation[1] = 0
        rotation[2] = 1
        rotation[3] = math.acos(cosAngle)

        length = math.sqrt(rotation[0] * rotation[0] + rotation[1] * rotation[1] +
rotation[2] * rotation[2])
        if length != 0:
                rotation[0] = rotation[0] / length
                rotation[1] = rotation[1] / length
                rotation[2] = rotation[2] / length
        if rotation[0] == 0 and rotation[1] == 0 and rotation[2] == 0:
                rotation[0] = 0
                rotation[1] = 0
                rotation[2] = 1
                rotation[3] = 0
        else:
                rotation[0] = 0
                rotation[1] = 0
                rotation[2] = 1
                rotation[3] = math.acos(cosAngle)
```

(b).wbt’s code change

```
        DEF REALWORLD_VEHICLE0 RangeRoverSportSVRSimple {
          translation -4 0 70
          rotation 0 -1 0 0.26179941
          name "vehicle(realworld)"
          controller "realworld"
          supervisor TRUE
    }
```

(c) About supervisor（Webots\projects\vehicles\protos）**Important!!!**

Select the type of car you use, right click the text file format to force the modification of the code, so that the Supervisor options and functions are available.

Example:LincolnMKZ.proto：

```
        PROTO LincolnMKZ [
          field     SFVec3f      translation        0 0.4 0
          field     SFRotation rotation             0 1 0 0
          field     SFColor      color                0.541 0.541 0.541
          field     MFString     plate                "textures/plate.jpg"
          field     SFString     engineSound          "sounds/engine.wav"
          field     SFString     name                 "vehicle"
          field     SFString     controller         "void"
          field     MFString     controllerArgs     []
          field     SFBool       supervisor           TRUE
          field     SFBool       synchronization    TRUE
          field     MFNode         sensorsSlotFront   []
          field     MFNode         sensorsSlotRear    []
          field     MFNode         sensorsSlotTop     []
          field     MFNode         sensorsSlotCenter []
          field     SFBool       frontSpotLights    FALSE
    ]

              name IS name
              model "Lincoln MKZ"
              controller IS controller
              controllerArgs IS controllerArgs
              supervisor IS supervisor
              synchronization IS synchronization
```

(d) Controller live transmission: **realworld.py**

```
        while supervisor.getBasicTimeStep() != -1:
            if s.data_rev:
                lat1 = s.data_rev['lat']
```

```
                    long1 = s.data_rev['lon']

                    x1, y1 = utm_coord_zero(long1, lat1)

                    ## calculate the coordiantes of realworld vehicle in Webots map
                    x = xoffest-x1
                    y = y1-yoffset
                    z = 0.5

                    POSITION = [x, y, z]
```

```
(While drivestep != -1
        WorldInfo basicTimeStep
        driver.getBasicTimeStep())  >1
```

The time in the simulation speed is greater than the time in the real world. Exe reads the transmitted data at a fixed interval. A formula transformation of real data corresponds to Webots update the location in WebotSD Basictimestep () for simulation time and display time)