1.1
Q1
- Thermometer: measures temperature, could be stored as double
- camera: measures visible lights, could be stored as 2d-array/matrix (of pixels)
- radar: measures the reflected pulse of energy, could be stored as tuple (wavelength, frequency)
- odometer: measures the distance traveled by a vehicle, could be stored as double
- LIDAR: measures pulsed laser (similar to radar), could be stored as tuple (wavelength, frequency)
- IMU: measures angular rate, could be stored as 1d-array (the length depends on the number of axises it measures)

Q2
sonar, infrared, touch sensor, GPS, compass

Q3
Robotic leg, shaft, headlight, brake

1.2
Q1
For a normal ellipse, the parametric equation looks like
$x(t) = 2\cos(t)$
$y(t) = \sin(t)$
Then, we rotate the coordinates for 30 degrees and get
$$x(t) = \sqrt{3}\cos(t) - \frac{1}{2}\sin(t) + 3$$
$$y(t) = \cos(t) + \frac{\sqrt{3}}{2}\sin(t) + 2$$
To find the control policy $\mathbf{u}(t)$, we need to find the derivative of the parametric equation shown above with respect to t. And we will get
$$\mathbf{u}(t) = [-\sqrt{3}\sin(t) - \frac{1}{2}\cos(t), -\sin(t) + \frac{\sqrt{3}}{2}\cos(t)]^T$$

Q2
For such an ∞-shape graph, its parametric equation looks like
$x(t) = \cos(t)$
$$y(t) = \frac{1}{2}\sin(2t)$$
Then, we rotate the coordinates for 45 degrees and get
$$x(t) = \frac{\sqrt{2}}{2}\cos(t) - \frac{\sqrt{2}}{4}\sin(2t)$$
$$y(t) = \frac{\sqrt{2}}{2}\cos(t) + \frac{\sqrt{2}}{4}\sin(2t)$$
Similar to Q1, we find the derivative with respect to t to obtain the control policy
$$\mathbf{u}(t) = [-\frac{\sqrt{2}}{2}\sin(t) - \frac{\sqrt{2}}{2}\cos(2t), -\frac{\sqrt{2}}{2}\sin(t) + \frac{\sqrt{2}}{2}\cos(2t)]^T$$

Q3
The helix curve is simply a circle on the xy-plane with a constant velocity in z-axis. The parametric equation looks like

$$x(t) = \cos(t)$$
$$y(t) = \sin(t)$$
$$z(t) = t$$

Then we can find the control policy by taking derivative

$$\mathbf{u}(t) = [-\sin(t), \cos(t), 1]^T$$

1.3
Q1
waypoints I chose:

$$p_0(-5,-7), p_1(10,-7), p_2(10,-3), p_3(4,-3), p_4(3,7), p_5(0,7), p_6(0,-3), p_7(3,-3), p_8(4,10), p_f(9,10)$$

The straight-line trajectories are

$$\gamma_{0,1}(t) = [1,0]^T t, \quad if\ t \in [0,15]$$
$$\gamma_{1,2}(t) = [0,1]^T t, \quad if\ t \in [15,19]$$
$$\gamma_{2,3}(t) = [-1,0]^T t, \quad if\ t \in [19,25]$$
$$\gamma_{3,4}(t) = [-0.1,1]^T t, \quad if\ t \in [25,35]$$
$$\gamma_{4,5}(t) = [-1,0]^T t, \quad if\ t \in [35,38]$$
$$\gamma_{5,6}(t) = [0,-1]^T t, \quad if\ t \in [38,48]$$
$$\gamma_{6,7}(t) = [1,0]^T t, \quad if\ t \in [48,51]$$
$$\gamma_{7,8}(t) = [0.1,1.3]^T t, \quad if\ t \in [51,61]$$
$$\gamma_{8,f}(t) = [1,0]^T t, \quad if\ t \in [61,66]$$

Q2
Similar to Q1 but with the point_to_point_traj() function. See 1.3.2.ipynb.

Q3
Firstly, I set the center of each red sphere as waypoints, and used the point_to_point_traj() and piecewise2D() to graph the trajectory. Then, with tests and simulations, I kept modifying the velocity at each waypoint to avoid collisions. Also, I added some necessary waypoints. The final result is shown in 1.3.3.ipynb. And here's the link of the simulation video:
https://drive.google.com/file/d/1goMLzoa9TO77zgb8cSSbPGMtakQSugNW/view?usp=sharing

Q4
Similar to the previous question, I first set all green spheres as waypoints. Then I modified point_to_point_traj() and piecewise2D() to let them generate 3D trajectories instead of 2D. I tested and modified the trajectory for several times, and some extra waypoints were added. The final result is shown in 1.3.4.ipynb. The link of the simulation video is:
https://drive.google.com/file/d/1zqwUnAVgeAT0C24sDviKtlQUiYAstcmv/view?usp=sharing