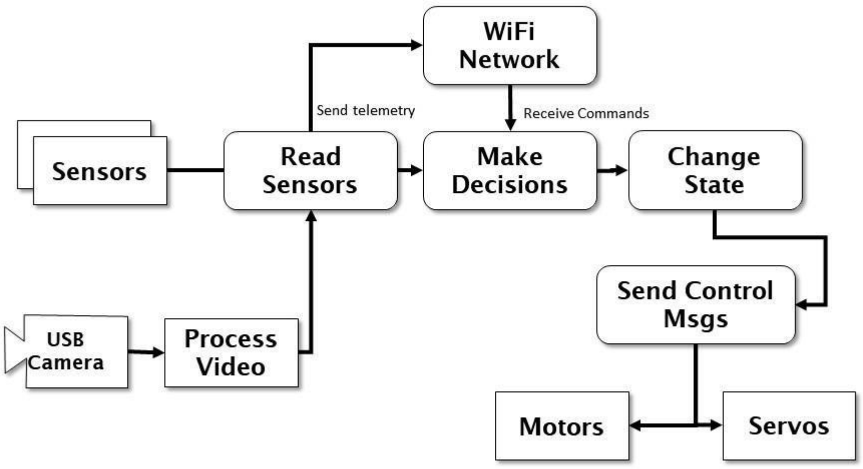
Robot control systems and a decision-making framework

Before we dive into the coding of our base control system, let’s talk about the theorywe will use to create a robust, modular, and flexible control system for robotics. As I mentioned previously, we are going to use two sets of tools in the sections: soft real-time control and the OODA loop. One gives us a base for controlling the robot easily and consistently, and the other provides a basis for all of the robot’s autonomy.

Soft real-time control

The basic concept of how a robot works, especially one that drives, is fairly simple.There is a master control loop that does the same thing over and over; it reads data from thesensors and motor controller, looks for commands from the operator (or the robot's autonomy functions), makes any changes to the state of the robot based on those commands,and then sends instructions to the motors or effectors to make the robot move:



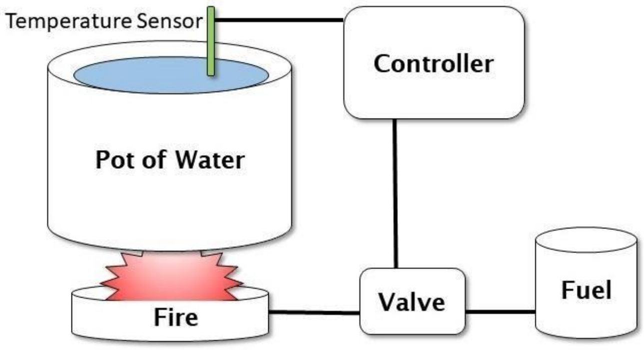
The preceding diagram illustrates how we have instantiated the OODA loop into thesoftware and hardware of our robot. The robot can either act autonomously, or accept commands from a control station connected via a wireless network.

What we need to do is perform this control loop in a consistent manner all of the time. We need to set a base frame rate, or basic update frequency, in our control loop. Thismakes all of the systems of the robot perform better. Without some sort of time manager, each control cycle of the robot takes a different amount of time to complete, and any sort ofpath planning, position estimate, or arm movement becomes more complicated.

If you have used a PID controller before to perform a process, such as driving the robot at a consistent speed, or aiming a camera at a moving target, then you will understandthat having even-time steps is important to getting good results.

Control loops

In order to have control of our robot, we have to establish some sort of control or feedback loop. Let’s say that we tell the robot to move 12 inches (30 cm) forward. The robothas to send a command to the motors to start moving forward, and then have some sort of mechanism to measure 12 inches of travel. We can use several means to accomplish this, butlet’s just use a clock. The robot moves 3 inches (7.5 cm) per second. We need the control loop to start the movement, and then at each update cycle, or time through the loop, check the time, and see if 4 seconds has elapsed. If it has, then it sends a stop command to the motors. The timer is the control, 4 seconds is the set point, and the motor is the system that is controlled. The process also generates an error signal that tells us what control to apply (in this case, to stop). The following diagram shows a simple control loop. What we want isa constant temperature in the pot of water:



The Valve controls the heat produced by the fire, which warms the pot of water. TheTemperature Sensor detects if the water is too cold, too hot, or just right. The Controller usesthis information to control the valve for more heat. This type of schema is called a closed loop control system.

You can think of this also in terms of a process. We start the process, and then get feedback to show our progress, so that we know when to stop or modify the process. We could be doing speed control, where we need the robot to move at a specific speed, or pointing control, where the robot aims or turns in a specific direction.

Let’s look at another example. We have a robot with a self-charging docking station,with a set of light emitting diodes (LEDs) on the top as an optical target. We want the robotto drive straight into the docking station. We use the camera to see the target LEDs on the docking station. The camera generates an error, which is the direction that the LEDs are seenin the camera. The distance between the LEDs also gives us a rough range to the dock. Let’ssay that the LEDs in the image are off to the left of center 50% and the distance is 3 feet (1m) We send that information to a control loop to the motors – turn right (opposite the image)a bit and drive forward a bit. We then check again, and the LEDs are closer to the center (40%) and the distance is a bit less (2.9 feet or 90 cm). Our error signal is a bit less, and the distance is a bit less, so we send a slower turn and a slower movement to the motors at this update cycle. We end up exactly in the center and come to zero speed just as we touch the
docking station. For those people currently saying "But if you use a PID controller …", yes,you are correct, I’ve just described a "P" or proportional control scheme. We can add morebells and whistles to help prevent the robot from overshooting or undershooting the target due to its own weight and inertia, and to damp out oscillations caused by those overshoots.

The point of these examples is to point out the concept of control in a system. Doing this consistently is the concept of real-time control.

In order to perform our control loop at a consistent time interval (or to use the properterm, deterministically), we have two ways of controlling our program execution: soft real time and hard real time.

A hard real-time system places requirements that a process executes inside a time window that is enforced by the operating system, which provides deterministic performance
– the process always takes exactly the same amount of time.

The problem we are faced with is that a computer running an operating system is constantly getting interrupted by other processes, running threads, switching contexts, and performing tasks. Your experience with desktop computers, or even smart phones, is that thesame process, like starting up a word processor program, always seems to take a different amount of time whenever you start it up, because the operating system is interrupting the task to do other things in your computer.

This sort of behavior is intolerable in a real-time system where we need to know in advance exactly how long a process will take down to the microsecond. You can easily imagine the problems if we created an autopilot for an airliner that, instead of managing theaircraft’s direction and altitude, was constantly getting interrupted by disk drive access or network calls that played havoc with the control loops giving you a smooth ride or making a touchdown on the runway.

A real-time operating system (RTOS) allows the programmers and developers to have complete control over when and how the processes are executing, and which routines are allowed to interrupt and for how long. Control loops in RTOS systems always take the exact same number of computer cycles (and thus time) every loop, which makes them reliable and dependable when the output is critical. It is important to know that in a hard real-time system, the hardware is enforcing timing constraints and making sure that the computer resources are available when they are needed.

We can actually do hard real time in an Arduino microcontroller, because it has no operating system and can only do one task at a time, or run only one program at a time. Wehave complete control over the timing of any executing program. Our robot will also have amore capable processor in the form of a Raspberry Pi 3 running Linux. This computer, whichhas some real power, does quite a number of tasks simultaneously to support the operating system, run the network interface, send graphics to the output HDMI port, provide a user interface, and even support multiple users.