Explanation of Concepts

StrangeMachine

A StrangeMachine manages the state for a particular component of the robot. They can be used in conjunction with one another to manage state changes involving multiple components, eg, the elevator and pickup positions.

A StrangeMachine has multiple "states", which represent the possible conditions the represented components can be in. You can `test` to see if a Machine is in a particular state, and `crank` a Machine toward a particular state; both operations will return whether or not the Machine is currently in the specified state.

RotationProvider

Given a PIDController that controls the turret rotation, a RotationProvider updates the setpoint of that controller, based on external feedback. This updating is done in the `update` method, which is called once per iteration in the main thread. While the PIDController is not, by default, enabled, this updating continues regardless.

Currently, the control structure works as follows:

- If the turret is in the `HIGH` position, then check if the user is pressing the `AIM_AND_FIRE` button.
- If so, make sure the PID controller is enabled and check if we're on target; if not, make sure the PID controller is disabled.
- If the `AIM_AND_FIRE` button is pressed, and we're on target, then go ahead and fire.

There are a few RotationProviders currently implemented. The one we've pretty much settled on is the `SlowbroRotationProvider`.

CameraInterface

A CameraInterface is an abstraction of a method the robot can use to obtain vision data. Currently, the only one is RemoteCameraTCP, which obtains data over a TCP connection, sent from processing software running on the Driver Station.

Springables

A Springable[Victor|Relay|DoubleSolenoid] works, in most ways, like a [Victor|Relay|DoubleSolenoid]. The only difference is the addition of a `reload` method. If the Springable[Victor|Relay|DoubleSolenoid] receives input, either through a PIDController or manually, it will put itself in the "sprung" state. When the `reload` method is called -- which it is for each Springable[Victor|Relay|DoubleSolenoid] at the end of the main control loop -- one of the following will happen:

- **if the Springable[Victor|Relay|DoubleSolenoid] is sprung:** un-spring it.
- **if the Springable[Victor|Relay|DoubleSolenoid] is not sprung:** set the output to the default output.

This way, if nothing writes to the Springable[Victor|Relay|DoubleSolenoid] over the course of a loop iteration, it will automatically switch itself off. This removes this burden from the main control logic, making things much, much simpler in implemenation.

Anything else?

Yep, there's a bunch more in here. Flip through the docs!