# Intro to Controllers

## Background

As mentioned in the previous document, we need to make sure that our robot doesn't break due to experiencing too much force. We found that reducing the change in velocity of the arm at any given moment to be less than or equal to 180 degrees per second would enable us to maximise the speed of our aim without accelerating too quickly and breaking the arm. In order to enforce this restriction, we are going to implement a proportional controller.

A proportional controller uses feedback from the current state of the system in order to control the next state of the system. The simple version will only have one variable that we can control (*V*). A proportional controller would measure the current value of this variable (*M*) and then compare it to a new target value of this variable (*N*) that we want to achieve. Rather than simply setting the value of the variable to be equal to the new value (*V* = *N*), we instead use information about the difference between the measured and target values (*M*-*N*) to decide what we should set the variable to.

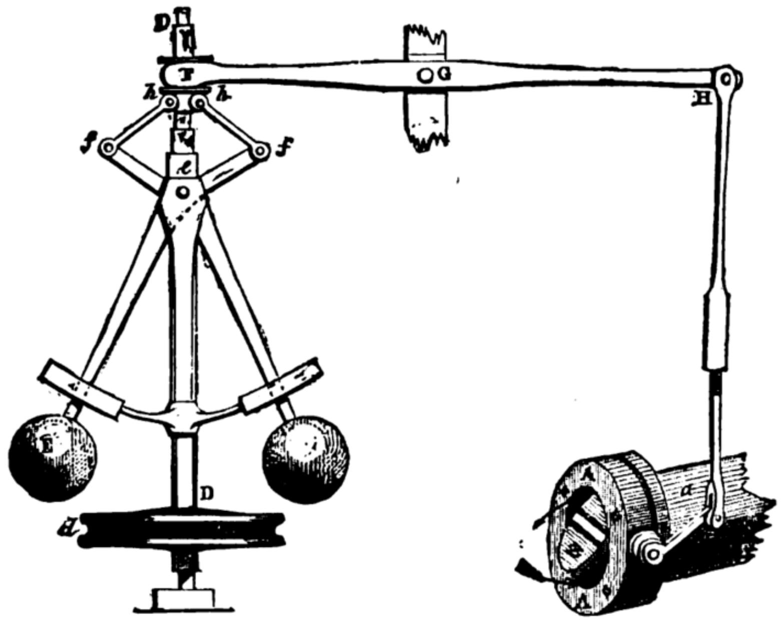


FIG. 4.—*Governor and Throttle-Valve.*

From Wikipedia: *The fly-ball governor is an early example of proportional control. The balls rise as speed increases, which closes the valve, reducing speed until a balance is achieved.*

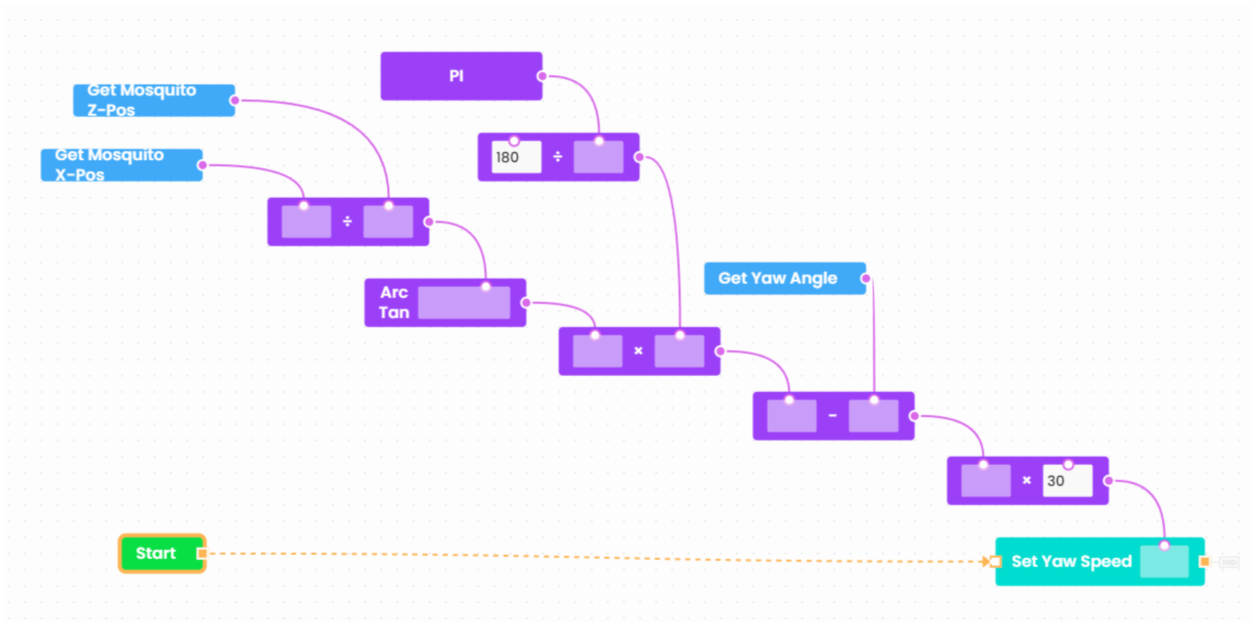
Why is this useful? Well, it can help reduce instability in systems with fast response times by stopping us from making changes to the system that are too sudden or large. If the difference between the current variable value and the target value is too large, then rather than setting *V* = *N*, we may instead want to move there slowly by halving the distance multiple times. If *M* is our current value and *N* is our target, then *M*-*N* will give us the difference between them. To move halfway between *M* and *N*, we can add half of this difference to *M*:

$V = M + 0.5(M - N)$

We can repeat this action multiple times, getting closer and closer to *N*. In our problem, implementing a controller will enable us to avoid accelerating too quickly and breaking the robot's arm.

## Our Problem

In our problem, our robot's yaw motor is always moving at a velocity *YM*. From our previous subsystems, we know how to calculate the yaw angle that our robot needs to reach in order to target the mosquito. In order to move our robot to this new angle, we set the velocity of this motor to quickly reach this location in time before the mosquito disappeared. You can see one way to set the speed of the motor below:



In this answer, the yaw velocity is set to thirty times the difference between the target yaw angle and the current yaw angle. This means that our robot will be turning faster the further away it is from the target and slower the closer that it gets. This helps us avoid overshooting. The problem with this answer, however, is that our robot ends up turning really fast when the target angle is far away from the current angle. This large change in velocity and resulting acceleration results in too much force and breaks the arm. You can test this out by copying your answer from subsystem 1 into subsystem 3 using the save and restore buttons in the bottom left corner of the dropzone.

To make sure that our arm never breaks but still moves as fast as possible, we will need to figure out three things:

1. How do we detect if the change of velocity is going to be too large?
2. If the change in velocity is going to be fine, how do we update our current velocity?
3. If the change in velocity is going to be too large, what do we change the velocity to instead?