Animatronics
Brain Document

By: Jim Schrempp and Bob Glicksman; v1.2, 1/08/2026

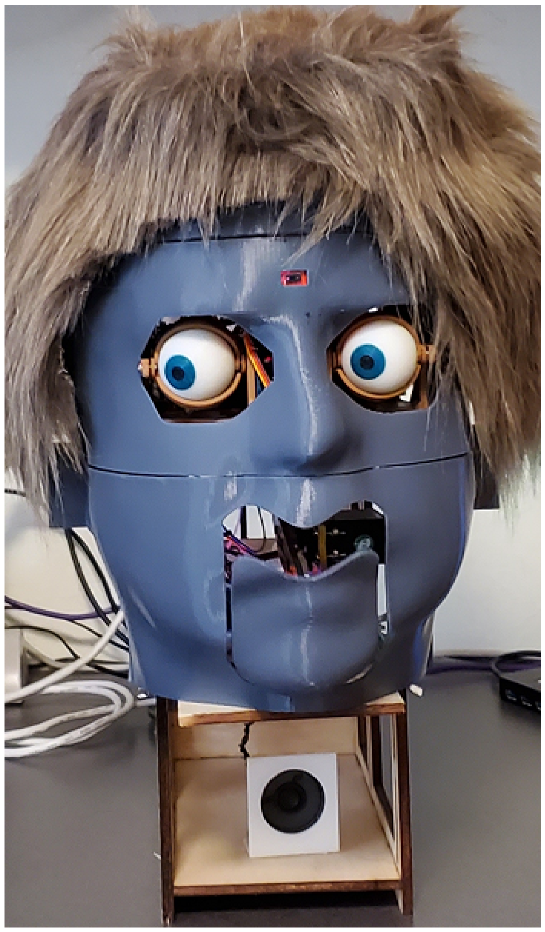

TABLE OF CONTENTS.

DOCUMENT OVERVIEW.

- Theory of Operation
    - How the software is structured
- 8x8 TOF Sensor
- Room Calibration
- Noise rejection
    - Spatial processing
    - Temporal processing
- Events
- Scenes and Scenarios
- Eye tracking based upon sensor data

THEORY OF OPERATION

Software Layout

The software to run the puppet is in:

https://github.com/TeamPracticalProjects/Animatronics/tree/main/Software/Photonfirmware/AnimatronicEyesTest

The main "brain" software is in AnimatronicEyes.ino and this is the place to start your learning. Other notable files in the src folder are:

- `TPP_Animatronic_Global.h` - Of interest is the `TOF_Detect` enumeration. This is a list of ways the puppet may interpret data from the ToF sensor, which then drives behavior.
- `TPPAnimateServo` - The class object `TPP_AnimateServo` that controls a servo. One instance per servo.
- `TPPAnimatePuppet` - The class object `TPP_Puppet` that controls coordination between servos to give a life-like look to the puppet's actions. This class would make sure that when the left upper eyelid closes, the lower left one closes too. One instance and it owns all the AnimateServo objects.
- `TPPAnimationList` - This class `animationList` is used to chain a series of "scenes" together over time. For instance, "GoToSleep" might have both eyes slowly close, then one eye open just a bit before it finally closes too.
- `AnimatronicEyes.ino` - The main program and the "brain" of the puppet. This code takes input from sensors (just the ToF sensor at this point), decides the emotional state of the puppet (bored, alert, asleep, etc), decides what - if any - actions to take based on the current emotional state and the current sensor input. This code owns the AnimationList and AnimatePuppet objects.

- `eyeservosettings.h` - Holds the constants for the particular eye mechanism discovered when you run the eye calibration software.

Software Layers

Starting at the lowest layer, each servo is controlled by an instance of the `TPP_AnimateServo` class. You call this class with a command to move to a certain position at a certain speed and the class object will make it happen in a non-blocking way.

The next layer up is one instance of the class `TPP_Puppet`. This class owns all the `TPP_AnimateServo` objects. It can handle coordinated movements. For example, we have a method to "blink". This requires the coordinated movement of four servos, one for each eyelid. In this method we might also have the focus of the eyes move just a bit to make the overall operation more life-like. If we had a servo to move the entire head left-right, then this would be the object to implement moving the eyes at the same time the head moves so that the eyes remain focused on one spot in space.

On top of `TPP_Puppet` we have `animationList`. A scene is a group of `TPP_Puppet` method calls. An `animationList` is a set of Scenes that happen over time. For example, one scene might be: Look Up and Left and blink. A second scene might be: Look right, close the eyes slowly, then quickly open the right eye only. The `animationList` object can be used to make these two scenes play out over time:

1. Starting immediately, move from the current positions to scene 1.
2. Starting in 10 seconds, move from the now current positions to scene 2.

By stringing together a series of well thought out scenes, complex animation sequences can be run to make the head appear life-like.

`animationList` provide the ability to have higher level concepts such as "go to sleep" or "wake up and look at the closest object" or (if we had neck movement) "wake up, quickly look at the nearest object and then slowly rotate the head so that it is facing the object head on".

main()	Calls animationList.process()
Class: animationList	A singleton. Define a series of "scenes" (mechanism positions) and then run through them as the clock ticks. ● addScene(scene, speed, delay) - predefined, or mechanism by mechanism ● startAnimation() ● stopAnimation() ● clearSceneList()

Class: TPP_Puppet	One for each puppet (we have only one); owns animateServo objects ● eyeball.look(x, y, speed) ● eyelid.open(position, speed) ● mouth.open(position, speed) ● Puppet ○ eyesOpen() ○ blink() ○ wink()
Class: TPP_AnimateServo	An instance for each servo. Moves servo slowly, or quickly. ● moveTo(x, speed)

The software stack

How it works.

1. AnimatronicEyes.ino
   a. Call TPP_TOF.getPOI or TPP_TOF.getPOITemporalFiltered(pointOfInterest *pPOI);
      i. Returns the current Point of Interest as determined by the TPP_TOF object. If the Temporal Filtered version is called, then it only reports a POI if the POI has persisted for a few seconds. This slows the puppet's response to new stimuli, but avoids reacting to spurious detections in the field of view.
      ii. Decide if this is a new POI and if so then how should the puppet react
      iii. Call for an animationList or directly command the TPP_Puppet
   b. Mainloop calls animationList.process()
   c. Of particular interest is `processEventsStateMachine` and the enum `headStates`. This is where the "brain" decides what action to take based on new sensor data and the current emotional state of the puppet.
2. animationList.process()
   a. notes elapsed time and decides if there is a new scene to set.
   b. Calls TPP_Puppet to set new scene positions.
   c. Calls TPP_Puppet.process()
3. TPP_Puppet.process()
   a. Decides if the movement of one servo affects another (e.g. moving eyes all the way left might cause the head to rotate left)
   b. Calls TPP_AnimateServo.process() for each servo it owns
4. TPP_AnimateServo.process()
   a. Notes current servo position vs target servo position
   b. Notes time and moves servo towards target position, if needed

TIME OF FLIGHT SENSOR (THE REAL "EYES")

The head uses an 8x8 optical Time of Flight sensor to detect objects in front of it. The ToF sensor provides an 8x8 matrix of values. Each value is the distance, in millimeters, to the closest object in that area of the field of view. If the closest object is more than about 8 feet away, then the sensor detects nothing.

When the puppet starts up it reads the current value of the ToF matrix and assumes that the distances it detects are all static objects in the environment. When the head first boots it is important that it is looking at a stable "no activity" field of view. After this the ToF sensor runs a new matrix 14 times a second. If our code is fast enough, it will consider each of the frames; if not we might miss one or two but that does not affect the operation of the "brain". On each read the brain assumes that deviations from the static background are objects it should be aware of. It is important that the head and background objects remain fixed or else the head will think objects have entered its field of view.

(We have plans, not implemented, to periodically re-sense the background objects. Our idea is that when the ToF values have not changed for 3 seconds, then the current ToF matrix should become the new background. The ToF is very sensitive, so a person standing still in front of it will always be seen to move slightly.)

For example, if nothing has changed from the start-up reading, the brain might decide that it is bored and put the eyes to sleep. Or it might decide to speak something like "Is any one out there?" while the eyes flit around a bit.

As another example, if the ToF data shows that something has entered the field of view the brain might decide to have the eyes "look" in that direction and the mouth to say, "hi there!"

The ToF sensor might detect several objects in the field of view. The brain has to sort these out and decide which ones to consider important. It favors ones that are closer and bigger over ones that are further away and smaller.

Noise Rejection

The ToF sensor is sensitive to even millimeter changes in distance. It is also subject to false readings due to multi-path distortion. For these reasons we have to do some filtering of the raw data.

Temporal filtering is done to make sure a new object is truly a new object and not just noise. The brain keeps several readings of the ToF sensor matrix. If a new object is only seen for a fleeting time, then we assume it was just noise from the ToF. Today a new object must be seen in two subsequent detection calls to be considered a Point of Interest.

Spatial filtering is done to determine which object is nearest. The brain considers adjoining cells in the 8x8 matrix. We average the ToF distance of adjacent cells to get the distance we will consider for a cell. We then consider the distance to each cell and direct the eyes to the cell with the nearest detection.

Eye Tracking

When the brain detects an object in its field of view it might decide to look at it. This means that the eyes must open and the eyes move vertically and horizontally to “look” at the object. As the object moves in the field of view of the TOF sensor, the brain may want to keep the eyes pointed at the object so that the eyes appear to track the object as it moves.

To make the movement appear more natural, the brain might have the eyes begin to move quickly towards the target, then slow as they approach their target position. Once in position, the eyes might have small saccade movements to improve the life-like feeling of the whole head. The eyes might blink from time to time. Blinking and saccade should be controlled below the "conscious" part of the brain, being a function of the eye class itself. The higher level "conscious" brain class should be able to instruct the eyes to “stare” and the eyes should not blink for a while, but eventually the eyes will need to blink.