Closed-Loop Systems

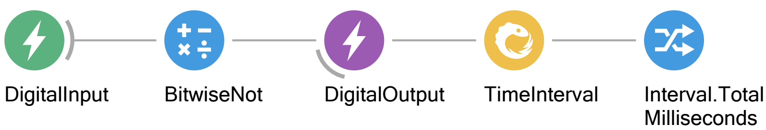
In a closed-loop experiment, we want the behaviour data to generate feedback in real-time into the external world, establishing a relationship where the output of the system depends on detected sensory input. Many behavioural experiments in neuroscience require some kind of closed-loop interaction between the subject and the experimental setup. The exercises below will show you how to use the online data processing capabilities of Bonsai to create and benchmark many different kinds of closed-loop systems.

Measuring closed-loop latency

One of the most important benchmarks to evaluate the performance of a closed-loop system is the latency, or the time it takes for a change in the output to be generated in response to a change in the input.

The easiest way to measure the latency of a closed-loop system is to use a digital feedback test. In this test, we measure a binary output from the closed-loop system and feed it directly into the input sensor. We then record a series of measurements where we change the output to `HIGH` if the sensor detects `LOW`, and change it to `LOW` if the sensor detects `HIGH`. The time interval between `HIGH` and `LOW` signals will give us the total closed-loop latency of the system, also known as the round-trip time.

Exercise 1: Measuring serial port communication latency

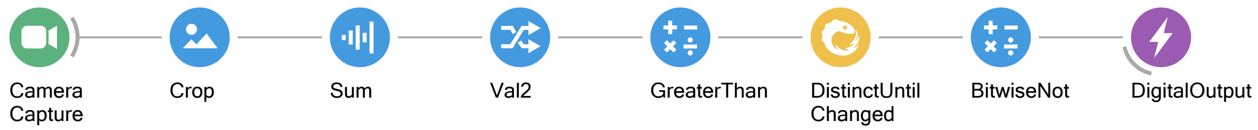


- Connect the digital pin 8 on the Arduino to digital pin 13 using a jumper wire.
- Insert a `DigitalInput` source and set its `Pin` property to 8.
- Insert a `BitwiseNot` transform.
- Insert a `DigitalOutput` sink and configure its `Pin` property to pin 13.
- Insert a `TimeInterval` operator.
- Right-click on the `TimeInterval` operator and select `Output` > `Interval` > `TotalMilliseconds`.

Note: The `TimeInterval` operator measures the interval between consecutive events in an observable sequence using the high-precision event timer (HPET) in the computer. The HPET has a frequency of sequence using the high-precision event timer (HPET) in the computer. The HPET has a frequency of at least 10MHz, allowing us to accurately time intervals with sub-microsecond precision.

- Run the workflow and measure the round-trip time between digital input messages.

Exercise 2: Measuring video acquisition latency



- Connect a red LED to Arduino digital pin 13.
- Insert a `CameraCapture` source.
- Insert a `Crop` transform.
- Run the workflow and set the `RegionOfInterest` property to a small area around the LED.

Hint: You can use the visual editor for an easier calibration. While the workflow is running, right-click on the `Crop` transform and select `Show Default Editor` from the context menu or click in the small button with ellipsis that appears when you select the `RegionOfInterest` property.

- Insert a `Sum (Dsp)` transform and select the `Val2` field from the output.

Note: The `Sum (Dsp)` operator adds the value of all the pixels in the image together, across all the color channels. Assuming the default BGR format, the result of summing all the pixels in the Red channel of the image will be stored in `Val2`. `Val0` and `Val1` would store the Blue and Green values, respectively. If you are using an LED with a color other than Red, please select the output field accordingly.

- Insert a `GreaterThan` transform.
- Insert a `BitwiseNot` transform.
- Insert a `DigitalOutput` sink and configure its `Pin` property to pin 13.
- Run the workflow and use the visualizer of the `Sum` operator to choose an appropriate threshold for `GreaterThan`. When connected to pin 13, the LED should flash a couple of times when the Arduino is first connected.
- Insert a `DistinctUntilChanged` operator after the `BitwiseNot` transform.

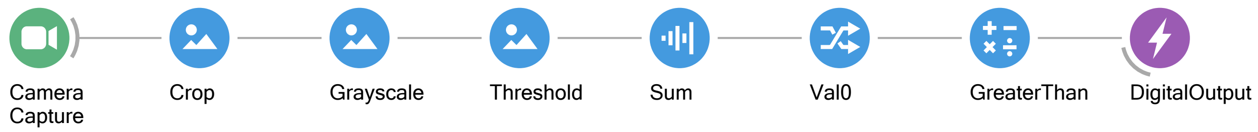
Note: The `DistinctUntilChanged` operator filters consecutive duplicate items from an observable sequence. In this case, we want to change the value of the LED only when the threshold output changes from `LOW` to `HIGH`, or vice-versa. This will let us measure correctly the latency between detecting a change in the input and measuring the response to that change.

- Insert the `TimeInterval` operator and select `Output` > `Interval` > `TotalMilliseconds`.
- Run the workflow and measure the round-trip time between LED triggers.

Given the measurements obtained in Exercise 2, what would you estimate is the ***input*** *latency for video acquisition?*

Closed-Loop Control

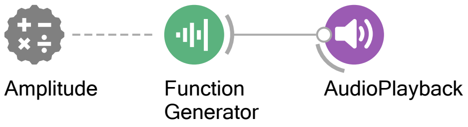
Exercise 3: Triggering a digital line based on region of interest activity



- Insert a `CameraCapture` source.
- Insert a `Crop` transform.
- Run the workflow and use the `RegionOfInterest` property to specify the desired area.
- Insert a `Grayscale` and a `Threshold (Vision)` transform (or the color segmentation operators).
- Insert a `Sum (Dsp)` transform, and select the `Val0` field from the output.
- Insert a `GreaterThan` transform and configure the `Value` property to an appropriate threshold. Remember you can use the visualizers to see what values are coming through the `Sum` and what the result of the `GreaterThan` operator is.
- Insert the Arduino `DigitalOutput` sink.
- Set the `Pin` property of the `DigitalOutput` operator to 13.
- Configure the `PortName` property.
- Run the workflow and verify that entering the region of interest triggers the Arduino LED.
- **Optional:** Replace the `Crop` transform by a `CropPolygon` to allow for non-rectangular regions.

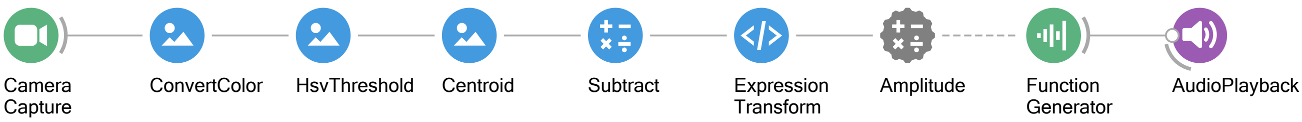
Note: The `CropPolygon` operator uses the `Regions` property to define multiple, possibly non-rectangular regions. The visual editor is similar to `Crop`, where you draw a rectangular box. However, in `CropPolygon` you can move the corners of the box by right-clicking *inside* the box and dragging the cursor to the new position. You can add new points by double-clicking with the left mouse button, and delete points by double-clicking with the right mouse button. You can delete regions by pressing the `Del` key and cycle through selected regions by pressing the `Tab` key.

Exercise 4: Modulating stimulus intensity based on distance to a point



- Insert a `FunctionGenerator` source.
- Set the `Amplitude` property to 500, and the `Frequency` property to `200`.
- Insert an `AudioPlayback` sink.
- Externalize the `Amplitude` property of the `FunctionGenerator` using the right-click context menu.

If you run the workflow, you should hear a pure tone coming through the speakers. The `FunctionGenerator` periodically emits buffered waveforms with values ranging between 0 and `Amplitude`, the shape of which changes the properties of the tone. For example, by changing the value of `Amplitude` you can make the sound loud or soft. The next step is to modulate the `Amplitude` property dynamically based on the distance of the object to a target.



- Create a video tracking workflow using `ConvertColor`, `HsvThreshold`, and the `Centroid` operator to directly compute the centre of mass of a colored object.
- Insert a `Subtract` transform and configure the `Value` property to be some target coordinate in the image.

The result of the `Subtract` operator will be a vector pointing from the target to the centroid of the largest object. The desired distance from the centroid to the target would be the length of that vector.

- Insert an `ExpressionTransform` operator. This node allows you to write small mathematical and logical expressions to transform input values.
- Right-click on the `ExpressionTransform` operator and select `Show Default Editor`. Set the expression to `Math.Sqrt(X*X + Y*Y)`.

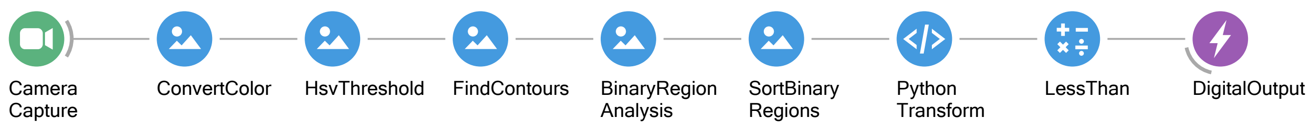
> **Note:** Inside the `Expression` editor you can access any field of the input by name. In this case `X` and `Y` represent the corresponding fields of the `Point2f` data type. You can check which fields are available by right-clicking the previous node. You can use all the normal arithmetical and logical operators as well as the mathematical functions available in the `Math` type. The default expression `it` means "input" and represents the input value itself.

- Connect the `ExpressionTransform` operator to the externalized `Amplitude` property.

- Run the workflow and verify that stimulus intensity is modulated by the distance of the object to the target point.
- **Optional:** Modulate the `Frequency` property instead of `Amplitude`.
- **Optional:** Use the `Rescale` operator to adjust the gain of the modulation by configuring the `Min`, `Max`, `RangeMax` and `RangeMin` properties. Set the `RescaleType` property to `Clamp` to restrict the output values to an allowed range.

> **Note:** You can specify inverse relationships using `Rescale` if you set the *maximum* input value to the `Min` property, and the *minimum* input value to the `Max` property. In this case, a small distance will generate a large output, and a large distance will produce a small output.

Exercise 5: Triggering a digital line based on distance between objects



- Reproduce the above object tracking workflow using `FindContours` and `BinaryRegionAnalysis`.
- Insert a `SortBinaryRegions` transform. This operator will sort the list of objects by area, in order of largest to smallest.

To calculate the distance between the two largest objects in every frame you will need to take into account some special cases. Specifically, there is the possibility that no object is detected, or that the two objects may be touching each other and will be detected as a single object. You can develop a new operator in order to perform this specific calculation.

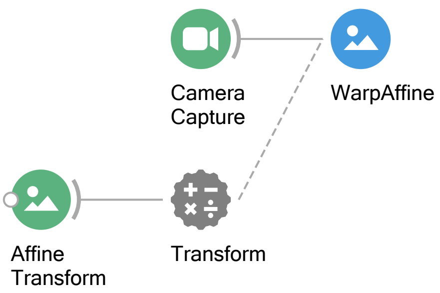
- Insert a `PythonTransform` operator. Change the `Script` property to the following code:

```
from math import sqrt

@returns(float)
def process(value):

  # no objects were detected
  if value.Count == 0:
    return float.NaN

  # only one object was detected, assume objects are touching
  elif value.Count == 1:
    return 0
```

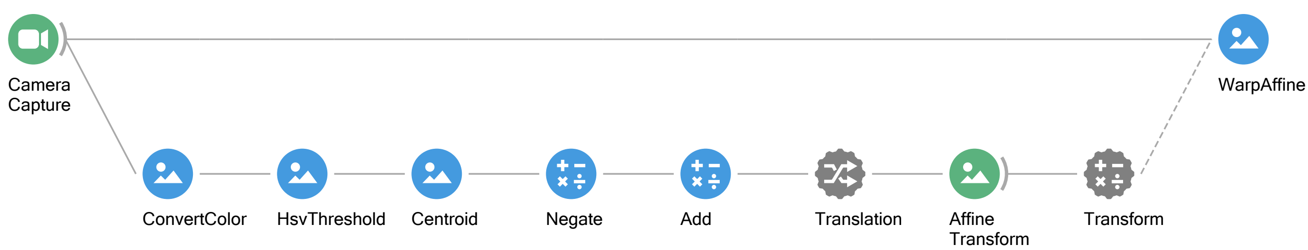
```
    # two or more objects were detected, compute distance
    else:
      # d: displacement between two largest objects
      d = value[0].Centroid - value[1].Centroid
      return sqrt(d.X * d.X + d.Y * d.Y)
```

- Insert a `LessThan` transform and configure the `Value` property to an appropriate threshold.
- Connect the boolean output to Arduino pin 13 using a `DigitalOutput` sink.
- Run the workflow and verify that the Arduino LED is triggered when the two objects are close together.

Exercise 6: Centring the video on a tracked object



- Insert a `CameraCapture` source.
- Insert a `WarpAffine` transform. This node applies affine transformations on the input defined by the `Transform` matrix.
- Externalize the `Transform` property of the `WarpAffine` operator using the right-click context menu.
- Create an `AffineTransform` source and connect it to the externalized property.
- Run the workflow and change the values of the `Translation` property while visualizing the output of `WarpAffine`. Notice that the transformation induces a translation in the input image controlled by the values in the property.



- In a new branch, create a video tracking pipeline using `ConvertColor`, `HsvThreshold`, and the `Centroid` operator to directly compute the centre of mass of a colored object.
- Insert a `Negate` transform. This will make the X and Y coordinates of the centroid negative.

We now want to map our negative centroid to the `Translation` property of

`AffineTransform`, so that we dynamically translate each frame using the negative position of the object. You can do this by using property mapping operators.

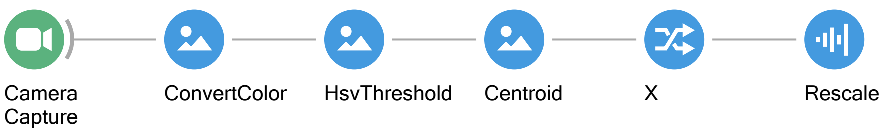
- Insert an `InputMapping` operator.
- Connect the `InputMapping` to the `AffineTransform` operator.
- Open the `PropertyMappings` editor and add a new mapping to the `Translation` property.
- Run the workflow. Verify the object is always placed at position (0,0). What is the problem?

> **Note:** Generally for image coordinates, (0,0) is at the top-left corner, and the center will be at coordinates (width/2, height/2), usually (320,240) for images with 640 x 480 resolution.

- Insert an `Add` transform. This will add a fixed offset to the point. Configure the `Value` property with an offset that will place the object at the image centre, e.g. (320,240).
- Run the workflow, and verify the output of `WarpAffine` is now a video which is always centred on the tracked object.
- **Optional**: Insert a `Crop` transform after `WarpAffine` to select a bounded region around the object.
- **Optional**: Modify the object tracking workflow to use `FindContours` and `BinaryRegionAnalysis`.

Exercise 7: Make a robotic camera follow a tracked object

On this exercise we will use the Pan and Tilt servo motor assembly to make the camera itself always point to the tracked object. The goal will be to keep the object always in the centre of the visual field of the camera. If the object is to the left of the centre, we turn the camera left, if it is to the right, we need to turn the camera right.

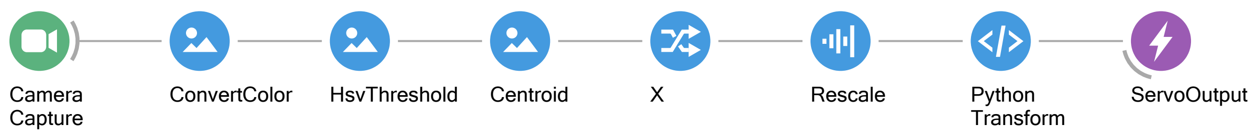


- Insert a `CameraCapture` source.
- Insert nodes to complete a video tracking workflow using `ConvertColor`, `HsvThreshold`, and the `Centroid` operator.
- Run the workflow and calibrate the threshold to make sure the colored object is perfectly segmented.

To make the Pan and Tilt servo motors correct the position of the camera, we now need to transform the X and Y values of the centroid, which are in image coordinates, to servo motor commands in degrees. For each frame we will have an incremental error depending on the observed location of the object, i.e. the deviation from the image centre.

- Right-click the `Centroid` and select `Output` > `X`.
- Insert a `Rescale` transform and set the `Max` property to 640 (the image width), and the `RangeMin` and `RangeMax` properties to 1 and -1, respectively.

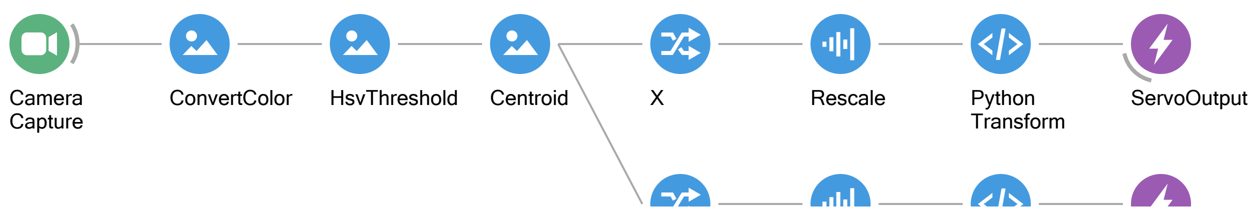
The output of this workflow will be a relative error signal indicating how much from the centre, and in which direction, the motor should turn. However, the commands to the servo are absolute motor positions in degrees. This means we will need to integrate the relative error signals to get the actual position where the servo should be. We also need to be aware of the servo operational range (0 to 180 degrees) in order not to damage the motors. To accomplish this, we will develop a new operator to compute the error-corrected integration before sending the final command to the servos.



- Insert a `PythonTransform` operator after `Rescale`. Change the `Script` property to the following code:

```
position = 90.0

@returns(float)
def process(value):
  global position
  temp = position + value

  # update the position only when the angle range is valid
  if (20.0 < temp and temp < 160):
     position = temp
  return position
```

- Insert a `ServoOutput` sink.
- Set the `Pin` property to the Arduino pin where the horizontal Pan motor is connected.
- Configure the `PortName` to the Arduino port where the micro-controller is connected.
- Run the workflow and validate the horizontal position of the motor is adjusted to keep the object in the middle.



- Right-click the `Centroid` and select `Output` > `Y` to create a new branch for the vertical Tilt motor.
- Insert a `Rescale` transform and set the `Max` property to 480 (the image height), and the `RangeMin` and `RangeMax` properties to -1 and 1, respectively (note these values are swapped from before because in image coordinates zero is at the image top).
- Copy and paste the `PythonTransform` script from the previous branch.
- Insert a `ServoOutput` sink and set the `Pin` property to the Arduino pin where the vertical Tilt motor is connected.
- Configure the `PortName` property.
- Run the workflow and validate the camera is tracking the object and keeping it in the centre of the image.

Visual Reactive Programming

A course on Visual Reactive Programming using Bonsai, developed by NeuroGEARS, Ltd.

neurogears
bonsai_rx

This website was prepared and developed for the Sainsbury Wellcome Centre, University College London.

Sainsbury Wellcome Centre