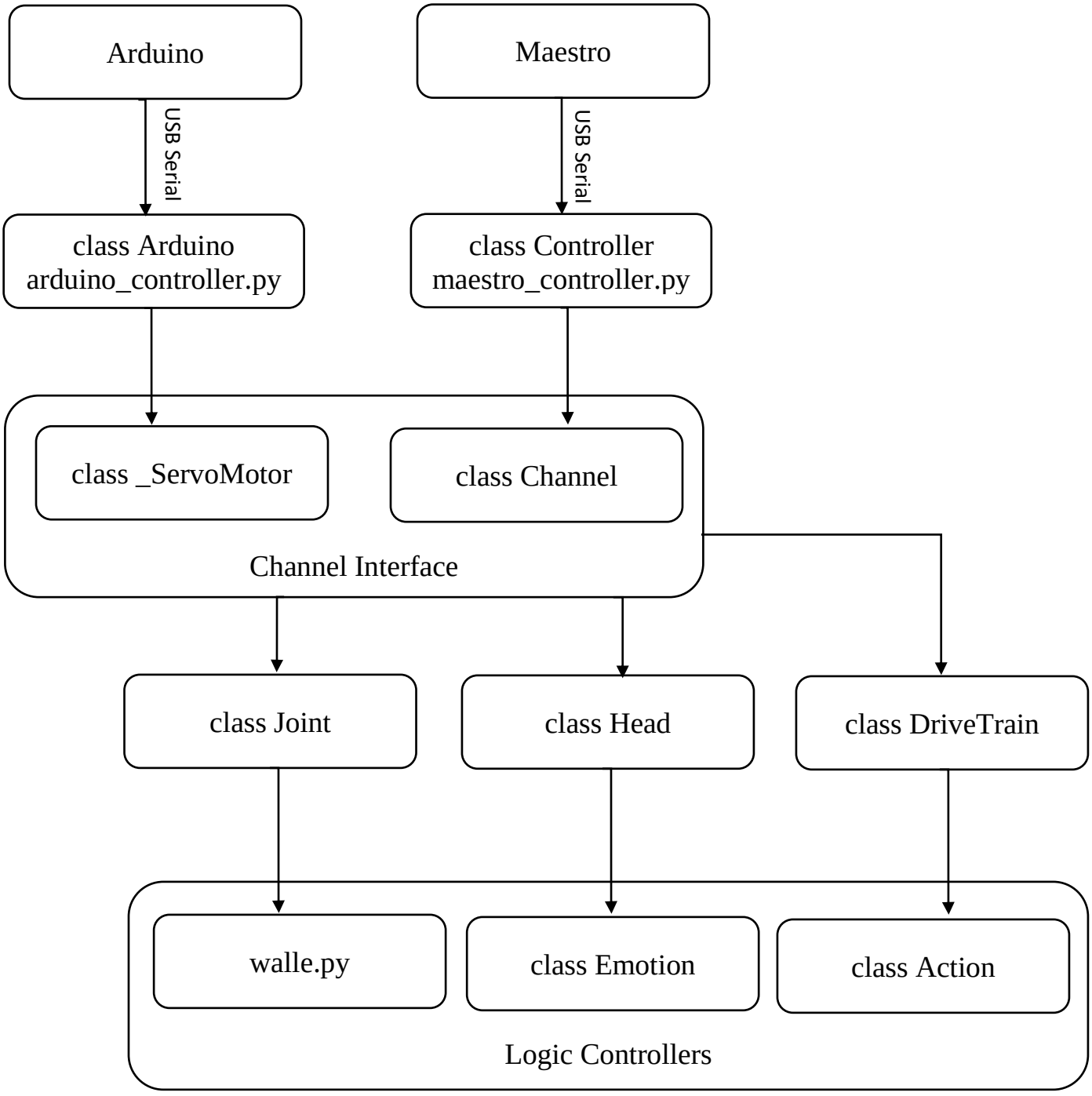
Wall-E Code Documentation

2018 Code Base Version

By Evan McCoy
Updated 14 July 2018

Project Abstraction Flow



Controllers and Channels

Wall-E uses two primary controller boards, an Arduino Nano and a micro maestro. Both boards communicate with the Python host over USB serial, but have very different protocols. To ease the use of these two controllers, there are two interfaces with mirrored function definitions that act as an abstraction layer between the arms and head functions and the controllers. Each control channel (i.e. a wrist, shoulder, or head axis) is defined from one of the two interfaces depending on its parent device. Arduino channels are defined as an instance of "_ServoMotor". Maestro channels are defined as an instance of "Channel".

Note: As currently written, it is up to the individual controller (Arduino/Maestro) to apply meaning to values as units. Ex. The maestro should take values between ~3-9,000, whereas the Arduino values are dependent on the interpretation by the Sketch in use. This should be noted when controlling positions/speeds for channels.

Interface Function Definitions

void setMin(int min) – Sets a minimum position to this channel.

void setMax(int max) – Sets a maximum position to this channel.

void setTarget(int position) – Sets a target position to this channel.

void setTargetSpeed(int speed) – Sets a target speed to this channel.

void setAcceleration(int acceleration) – Sets a target acceleration to this channel.

int getPosition() – Gets the current position for this channel. May block the thread for a ***short*** period while waiting for a response over serial.

Channel Interface Usage

Arduino:

```
from controllers import controllers

channelController = controllers.arduino.getMotor(channel)
```

Maestro:

```
from controllers import controllers

controllerChannel = controllers.maestro.Channel(channel)
```

The Joint Class

The Joint class represents a single joint or axis of motion in the arms. Joints are constructed from a controller channel and two optional arguments. A Joint instance allows you to move a joint to a position at a specified speed, check if the joint is within its limit of motion, and check if the joint is moving toward a target position or completed its target.

Joint Function Definitions

Joint(int channel, int[] limits=[-sys.maxint, sys.maxint], bool maestro=False)

Params:

int channel – *The channel index on the parent controller.*

int[] limits – *Optional input limits on the range of motion.*

bool maestro – *Optional controller spec. If true, a maestro channel will be used. If false, an Arduino channel will be used.*

bool moveable() – Whether or not the joint is at or beyond its limits.

bool moving() – Whether or not the joint has yet to reach its defined target.

bool complete() – Whether or not the joint has reached its defined target. !moving()

void movePos(int position, speed=0) – Moves to a target position with an optional speed parameter.

The Arms Class

The arms class is a container which holds the four associated joints of Wall-E's arms.

Arms Members

Joint left_shoulder

Joint right_shoulder

Joint left_hand

Joint right_hand

The Emotions Mechanism

The emotions mechanism is composed of an Emotion class which stores a set of Actions. These actions can be triggered by a timestamp or by a provided trigger function. The emotion will run until all child actions have been completed. Emotions are designed so they can be passed to higher emotions and will operate as an Action.

Emotion/Action Function Definitions

void run() – Runs the given Emotion or Action. Should be executed continuously via a parent Emotion's update() function. Emotion#run may only run once, but may safely be run continuously if being treated as an Action.

bool complete() – Whether or not the action has reached its goal or emotion has completed all child actions.

void reset() – Resets the complete flag for Actions, does nothing for Emotions.

Emotion Function Definitions

Emotion(long time=None, func trigger=None) – Constructs an emotion with no actions. Optional timestamp (from the beginning of the parent emotion) or boolean-return trigger function inputs for use as an Action in a parent Emotion.

void addAction(Action action) – Adds an action to this emotion.

void update() – Should be run continuously. Manages the triggering of actions and runtime environment of the emotion.

bool isRunning() – Whether or not the Emotion is currently running.

Action Function Definitions

Action(func run, long time=None, func trigger=None) – Constructs an action with the defined run function. Must define either a start time (from the beginning of the parent Emotion) or a trigger function which returns a boolean.