ASIP 1.1 Changes

ASIP version 1.1 has a number of changes to the source code to support the following functional enhancements:

- Support for motors other than HUBee wheels
- New motor functions to control robot speed and rotation using PID. These functions are suitable to ROS enable a robot. Note the PID functions are only implemented for the new motors.
- The ASIP core library now also includes services such as: Robot, Neopixel, sound and other services so a single ASIP library install brings in all commonly used services.

Notes when testing with the HUBee wheel Mirto robot

- Encoder events enabled with: "M,A,1\n" , disable with "M,A,0\n"
- PID functions will not work well with HUBee (insufficient pulses per wheel revolution)

If using sketches other than the Mirto2016 example in the repository make the following changes:

- Encoder service must be removed from sketch (it's part of Motor service)
- Path names for service header files changed from <utility/foo.h> to <service/foo.h>

Notes when testing with the new motors and PCB

- #define MIRTO2016 in robot_pins.h **should NOT** be commented
- #define HUBEE_WHEELS **should be** commented out in RobotDescription.h

Backward compatibility

With one exception, existing ASIP clients should run unaltered with the new version, encoder events in this version is handled differently. The Encoder event interval is now hard coded within the motor service module. Encoder events can be turned on and off using ASIP protocol but the interval cannot be changed (without recompiling the motor source code). Encoder events are now part of the motor service, therefor the encoder event now uses the motor service ID 'M' rather than 'E'. The message fields have also changed to contain the number of pulses since the last message and the total pulse count. Previously the fields were pulse width in microseconds and number of pulses since last message. A new message has been added to the protocol to enable a client to reset the total pulse count to zero.

For example, here is the previous encoder message to request encoder data every 20 milliseconds:

```
Request:"E,A,20\n"
Reply:  "@E,e,2,{3000:11,3100:12}\n"
```

In this example, pulse widths are 3000 & 3100 microseconds, step counts are 11 & 12.

Is replaced by the following:

request encoder data (at an interval set by the server, typically 33ms)

```
Request:"M,A,1\n"   note:  "M,A,0\n" would turn off events
Reply:  "@M,e,2,{11:234,12:237}\n"
```

In this example, step counts are 11 & 12, total counts are 234 and 237.

Bear in mind when updating client code that the number of pulses per revolution of the wheel (PPR) with the new motors is much higher than with the HUBee wheels (new motors PPR= 1632, the HUBee PPR=64).

Other than the encoder messages, all existing ASIP client code running with version 1.1 should function the same as with 1.0.

New commands

In addition to the updated Encoder handling mentioned above, new commands have been added for enhanced motor control.

SetMotorRPM()
Description: Controls motor RPM using PID
API Syntax: setMotorRPM(motor, RPM, duration);
Parameters:
  motor is wheel id (0 or 1)
  RPM (-x to x TODO)
  duration in milliseconds
Example set motor 1 RPM to 30 for 2 seconds: setMotorRPM(30,20000);
Stream protocol for this example: "M,r,1,30,2000\n"
Note: motor rpm can be changed or motor stopped prior to duration timeout (function is non-blocking)

SetMotorsRPM()
Description: Controls both motors RPM
API Syntax: setMotors(power0, power1);
Parameters:
  RPM0 (-x to x TODO)
  RPM1 (-x to x TODO)
  duration in milliseconds
Example set both motors RPM to 30 for 1 second: setMotorsRPM(30,30,1000);
Stream protocol for this example: "M,R,30,30,1000\n"

setRobotSpeedCmPerSec()
Description: sets to speed of the robot in cm per second
API Syntax: setRobotSpeedCmPerSec(cmps, duration);
Parameters:
  cmps speed in centimeters per second
  duration in milliseconds
Example move robot backward at 20 cm per sec for 2 seconds: setRobotSpeedCmPerSec(-20,2000);
Stream protocol for this example: "M,c,-20,2000\n"

rotateRobotAngle()

Description: rotates robot at the given speed and angle
API Syntax: rotateRobotAngle (degreesPerSec, angle);
Parameters:
  degreesPerSec
  angle to rotate in degrees
Example rotate 180 degrees in 2 seconds: rotateRobotAngle(180, 90);
Stream protocol for this example: “M,a,180,90\n”

RequestEncoders()
Description: Enables or disables repeated reading and sending of encoder counts for all encoders
API Syntax: RequestEncoders(enable)
Parameters: 0 disable, any other value enables
Example request encoder data
Stream protocol: “M,A,1\n” note that this has changed from previous version
Example turn off scheduled messages:
Stream protocol: “M,A,0\n”

encoderReply msgs
Description: informs count and duration for all encoders
Syntax: reply(,nbr,pulse0,count0, pulse1, count1)”
Parameters:
  nbr is the number of encoders (currently 2)
  pulse0 is the count since the previous message for the first encoder
  count0 is the total count value for the first encoder
  pulse1 is the count since the previous message for the second encoder
  count1 is the total count value for the second encoder

Reply message protocol is: @M,e,2,{0:0,0:0}\n

Building ASIP servers

ASIP library structure and installation

Version 1.1 leverages improved dependency handling in recent Arduino releases to enable a single library package to combine services into a single folder. Therefore Arduino 1.6.8 or later must be used (this code has been tested using 1.6.9 and 1.6.11).
All required libraries are included with the Teensyduino install (todo – needs to be checked with a fresh install).
The following libraries are required for other processors
Encoder
(todo other dependencies needs to be checked with a fresh install).

Libraries not used by the sketch no longer need to be included (earlier Arduino compilers required the inclusion within the sketch of any library that was referred to in a service even if that service was not used in the sketch.

ASIP server sketches

The following are best practice for sketches using ASIP 1.1

- asipIO.begin() method should follow all other service begin methods to ensure that the core IO is aware of pin usage by all other services.
- robot_pins.h and HUBeeWheel.h no longer need to be included in the sketch. These are now handled within robot.cpp
- explicit #include of external libraries such as Servo.h and Neopixel.h are no longer required