SATS Beaglebone Black ROS

April 14, 2016

1 sats_car_ros operation

Operation of the ros interface for the sats cars is simple. The following commands are all that are needed to run the cars:

```
roslaunch sats_car_ros run_cars.launch
```

A gui is available by opening rqt from another window.

1.1 Parameters

The run_cars.launch file references two parameter files: sats_car_ros/yaml/params.yaml Listing **??**, sats_car_ros/yaml/bbx_mot
Listing **??**, where bbx represents the number of the car being used. These parameters are automatically loaded, but they can be re-broadcast using the following command:

```
rosparam load ~/path/to/yaml/params.yaml /params
rosparam load ~/path/to/yaml/bbx_motors.yaml /bbx/motor_gains
```

Listing 1: 'parameters set in params.yaml'

```
kp: 1.0
kd: 2.0
h : 0.5
nom_dist: 0.5
kpATK:  3.0
kdATK: -1.0
hATK :  0.5

L: 0.31  # vehicle width

controller_rate:  50.
sensor_rate:      70.
high_level_rate:  5

ll_kp: 1.35
ll_ki: 6.25
ll_kd: 0.0008
pwm_sat: 0.5 # pwm saturation limit. set to 0.95 during performance mode
```

Listing 2: 'parameters set in bbx_motor.yaml'

```
aL: -2.74081632653
aR: -2.74081632653
```

```
bL: 0.801873682274
bR: 0.807646304422
deadbandL: 0.1475
deadbandR: 0.1475
gps_mode: True
```

1.2 Message to operate cars

In rqt, there is a message publisher. This is what we will use to operate the robots. The two messages are master Listing **??** and car_command Listing **??**.

Listing 3: 'MasterCommand message definition'

```
# Message that transports user commands.

# the header contains basic timestamp info
std_msgs/Header header
# manual activates / deactivates the high_level controller
bool master_on
# update parameters on all vehicles
bool update_param
```

Listing 4: 'CarCommand message definition'

```
# this defines the structure of CarCommand. Do not use this directly
#
std_msgs/Header header
bool HL_active
bool HL_atk_mode

bool LL_active

# Vel mode. Set a velocity with an amplitude and period.
# Vel mode is only active if HL and LL are active
bool VEL_mode

float32 command_velocity
float32 d_amplitude
float32 d_period # 0 means velocity is constant

# Curvature automatic
bool curv_auto
float32 curv


# PWM mode is used for characterization. Use velocity mode for normal
   operation.
# PWM mode is given priority over the HL and LL
bool PWM_mode
# pwm values are used only if pwm mode is set
float32 pwmL
float32 pwmR
```

2 Matlab message extraction

If you are using MATLAB 2015a or later, support for ROS is built in, but we use custom message types. To load the message types into your path, use the instructions from MATLAB.

Listing 5: method for using custom messages in matlab

```
To use the custom messages, follow these steps:

1. Edit javaclasspath.txt, add the following file locations as new lines,
   and save the file:

/path/to/svn/code/ros/matlab_gen/jar/sats_car_ros-1.0.1.jar
/path/to/svn/code/ros/matlab_gen/jar/pixy_msgs-1.0.0.jar
/path/to/svn/code/ros/matlab_gen/jar/pixy_node-1.0.0.jar
/path/to/svn/code/ros/matlab_gen/jar/pixy_ros-1.0.0.jar

2. Add the custom message folder to the MATLAB path by executing:

addpath('/path/to/svn/code/ros/matlab_gen/msggen')
savepath

3. Restart MATLAB and verify that you can use the custom messages.
   Type "rosmsg list" and ensure that the output contains the generated
   custom message types.
```

3 Appendix

3.1 Copy EEMC to SD

- Boot from EEMC by powering the device on.
- Copy image from EEMC to SD.

  ```
      dd if=/dev/mmcblk0 of=/dev/mmcblk1 bs=1M
  ```

- Boot from SD. Press S2 (near SD card) while booting.
- Copy image from SD to EEMC. Note that when booted from the SD card, the address of the EEMC is now mmcblk1.

  ```
      dd if=/dev/mmcblk0 of=/dev/mmcblk1 bs=1M
  ```

3.2 Programs to install on clean image

1. Obtain a clean ubuntu image for BBB
2. Install the following:
   - openssh-server
   - subversion
   - python2.7
   - python-numpy

- ros-*version*-base
- ros-*version*-tf
- ros-*version*-angles
- libusb-1.0.0-dev
- python-serial

3.3 OpenCV Configuration

OpenCV must be cross-compiled for ARM with NEON and VFPV3 hardware acceleration enabled. For completeness sake, we also install the opencv_contrib setup to enable SURF and SIFT algorithms. Some tutorials on how to do this are:

- `http://vuanhtung.blogspot.com/2014/04/and-updated-guide-to-get-hardware.html`
- `http://docs.opencv.org/2.4/doc/tutorials/introduction/crosscompilation/arm_crosscompile_with_cmake.html`
- `http://blog.lemoneerlabs.com/3rdParty/Darling_BBB_30fps_DRAFT.html`
- `https://github.com/itseez/opencv_contrib`

I couldn't get python support through cross compiling, but when I compiled on the BBB, the python support worked. The micron USB contains the build files. Mount it at /media/usb/ to install.

- navigate to /media/usb/opencv/opencv-3.1.0/build
- sudo make install

3.4 Enabling ports

The BBB utilizes device trees to enable peripheral and show the address to the peripheral. A great resource with sample overlays is located at `https://github.com/beagleboard/bb.org-overlays`.

Source files *.dts are compiled to *.dtbo binaries using dtc. `https://learn.adafruit.com/introduction-to-the-beaglebo` `compiling-an-overlay`. The source files define the port muxing and such. The ports used in this project are shown in Table **??**.

$Pins	Header	Pin	Name	Mode
28	9	11	UART4-RX	6
29	9	13	UART4-TX	6
18	9	14	PWM1A	6
19	9	16	PWM1B	6
87	9	17	I2C1-SCL	2
86	9	18	I2C1-SDA	2
85	9	21	UART2-TX	1
84	9	22	UART2-RX	1
96	9	26	UART1-RX	0
97	9	24	UART1-TX	0
48	8	37	UART5-TX	4
49	8	38	UART5-RX	4

Table 1: The configuration of the peripherals we're using

Table 2: The method for initializing the ports that will be used.

```
export SLOTS=/sys/devices/platform/bone_capemgr/slots
echo BB-PWM1 > $SLOTS
echo BB-UART1 > $SLOTS
echo BB-UART2 > $SLOTS
echo BB-UART4 > $SLOTS
#echo BB-I2C1 > $SLOTS  #I haven't tested if this I/O actually works, but
   it might.
export PERIOD=20000000  #This period is nanoseconds, and the motors need a
    period of 20 ms

sleep 1 # to allow the device tree time to set in place.

export PWM=/sys/class/pwm/pwmchip0

echo 0 > $PWM/export
echo 1 > $PWM/export
echo $PERIOD > $PWM/pwm0/period
echo 1 > $PWM/pwm0/enable

echo $PERIOD > $PWM/pwm1/period
echo 1 > $PWM/pwm1/enable
chown -R sats:sats /sys/class/pwm/pwmchip0/pwm*

# access pwm0 and pwm1 by file /sys/class/pwm/pwmchip0/pwm0/duty_cycle,  /
   sys/class/pwm/pwmchip0/pwm1/duty_cycle
# UART is accesible in /dev/ttyO*, where the O is not a zero, and * is the
    UART port.
# Tiva C is wired to UART4 on Beaglebone, communitcating through UART1 on
   Tiva C.
```

The *.dtbo files are stored in /lib/firmware/ which then can be called by writing the file to /sys/devices/-platform/bone_capemgr/slots. The initialization file to be used is shown in Table ??.

3.5 ROS Workspace Configuration

- Create workspace directory, /catkin_ws
- Create source directory, /catkin_ws/src
- Navigate to src, initialize workspace: catkin_init_workspace
- Place the following packages in the /catkin_ws/src directory:
  - sats_ros_car: svn co `svn+ssh://user@macbeth.think.usu.edu/svn/aet/code/sats_car_ros`
  - pixy_ros: git clone `https://github.com/jeskesen/pixy_ros`. The headers need to be moved from the devel directory or included from there.
- run catkin_make in /catkin_ws/
- If desired, add /catkin_ws/devel/setup.bash to the .bash_rc file

Table 4: The commands to enable wifi on the BBB.

```
ifconfig wlan0 up
iwconfig essid ``DroneNet''
ifconfig wlan0 inet 192.168.1.2 netmask 255.255.255.0
```

3.6 Network Configuration

3.6.1 Hosts

Change the /etc/hostname file to desired name, bbb0 to bbb9.

Add the following to /etc/hosts:

Table 3: Contents of /etc/hosts file to be used to enable networking

```
127.0.0.1 localhost
127.0.1.1 bbb1.localdomain

192.168.1.100 bbb0
192.168.1.101 bbb1
192.168.1.102 bbb2
192.168.1.103 bbb3
192.168.1.104 bbb4
192.168.1.105 bbb5
192.168.1.106 bbb6
192.168.1.107 bbb7
192.168.1.108 bbb8
192.168.1.109 bbb9
192.168.1.200 sats-desktop
```

3.6.2 Enable wifi

Enable wlan0, connect to DroneNet, set ip address to the corresponding address from Table **??**. Example in Table **??**.