COMP3431

Robotic Software Architecture

Assignment 2: Report

Aaron Ramshaw
Simon Robilliard
Lucy Kidd
Wayne Yeung

November 3, 2015

Introduction

The project required the development of an interface between a Raspberry Pi v2, Dynamixel motors and a Raspberry Pi Camera Module to form a mini EMU Robot. Software would be developed using Robot Operating System (ROS) such that the EMU could be controlled remotely using an xbox controller and an image stream. The mini EMU consists of a rectangular chassis with four wheels connected to its two longer outer edges. A long arm is mounted, via a motor to the upper carriage of the chassis, and two motors atop the arm create two more pivot points, with the top motor holding the Raspberry Pi Camera. An already constructed mini EMU can be seen in Figure 1.1.

Background

Robot Operating System (ROS) is an open-source, meta-operating system for use with robots. It provides a range useful services including: hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management (ROS wiki, 2014). Within the ROS community there are open-source packages available for a range of sensors and drivers, this includes the Raspberry Pi camera and Dynamixel Motors. ROS also provides tools to run code across multiple computers at once, hence provides the perfect software foundations for this project.

The Raspberry Pi is advertised as a credit card sized computer which can be plugged into a display via a High-Definition Multimedia Interface (HDMI) cable and uses a standard USB keyboard and mouse. The tiny computer consists of a 900 MHz quad core Central Processing Unit, a 1GB Synchronous Dynamic Random-Access Memory (SDRAM) and an Operating System(OS) of user choice that is stored as a Disk Image on a removable micro-Secure Digital(SD) card. There are 30 plus General Purpose Input Output (GPIO) pins, four USB input plugs and a 30 way camera connector, which makes the Raspberry Pi perfect for Robotics applications.

The Dynamixel AX-12A servo actuator from the Korean manufacturing company Robitis, has the ability to track its own speed, temperature, shaft position, voltage and load. The shaft position of of the ax-12 actuator can be adjusted individually for each motor, allowing a different speed and strength for each motor (Robitis, 2010). Each motor can be initialised as either a standard servo motor or as a continuous rotation servo, hence the motor is applicable for use as the EMU robots wheel drivers and arm pivot points.

The Raspberry Pi Camera Module is a small camera that connects to a Raspberry Pi via a 30 way ribbon cable. The Raspbian OS has built in drivers for cameras connected via this connector, and an Open-Source Application Program Interface (API) for C++ available for use (Salinas, R. 2015). The camera has video modes of up to 1080p at 30 frames per second (fps) and a still frame resolution of up to 5 Megapixels, hence provides potential for a high quality video stream.

Chassis and Hardware

Software

Architecture

Debian based Raspbian is the official supported operating system for the Raspberry Pi. The OS is installed to the microSD card of the Pi. Raspbian was chosen because of its ease of installation and full support and compatibility with the Pi. ROS Indigo is the software platform used for the robot.

Dependencies

The project makes use of the Dynamixel Controllers package - the low level drivers for the Dynamixel motors. The Joystick Drivers package is used to receive commands from the Logitech controller. In addition, Raspicam drivers are used to interface with the Raspberry Pi camera. The four nodes in our package are Commander, Emu State Publisher, Image Publisher and Joystick Converter.

Commander

This node is responsible for subscribing to the the topics published by Joystick Converter for controller input and motor state from Emu State Publisher. It then publishes appropriate commands to the 7 motor controllers.

The four wheel motors are set to pan mode which allows us to specify speeds, published to topics `/pan_controller_[1-4]/command` which take type `Float64` messages. The three arm motors are set to servo tilt mode which allows us to specify an angle, published to topics `/tilt_controller_[1-3]/command`, which

also take type `Float64` messages (angle in radians).

The `joy_stick_converter` node publishes to two topics `motor_command_vector` and `servo_command_vector`. These two topics are `TwistStamped` messages, containing 2 3D vectors `linear` and `angular`.

For `motor_command_vector` we are only concerned with the `x` and `y` values of the `twist.linear` vector. This is the direction vector for the movement of the robot. We split up the components into left and right movement, then publish the speed `Float64` value to the left and right side motors respectively. These four motors controlling the wheels are set to pan mode, which allows us define a speed to run at.

Similarly, the node listens to `servo_command_vector`, which gives directions regarding the movement of the 3 motors making up the arm. Here we consider both the `twist.linear` and `twist.angular` vectors. The values of the linear vector are used to increment the position of the arm (all 3 `x`, `y` and `z` for the 3 motors) by publishing new incremented positions to `/tilt_controller_[1-3]/command` after reading the current positions from `/servo_controller_[1-3]/state`. The `twist.angular` vector is read to set the arm motors into set default positions. Instead the given angles are simply published to the `/tilt_controller_[1-3]/command` topics.

Due to the physical constraints of the chassis, limits are set on the arm motor angles to prevent pushing on the frame and pulling on cables. These limits are taken from the parameters server at launch and override the `/tilt_controller_[1-3]/command` values when appropriate. The middle joint motor is also artifically limited when the arm is extended past the base at an angle greater than 90 degrees - a position which uses the arm to push the whole robot chassis up. This retracts the camera joint, preventing the more fragile camera module from being damaged.

In addition, to facilitate easy and natural movement from the perspective of the robot driver (using the camera stream), the middle motor is calibrated with the arm base motor such that vertical camera angle remains constant when the arm base motor is moved. This is done by moving the middle joint motor in the opposite direction to the base joint.

Emu State Publisher

This node subscribes to the topics `/tilt_controller_[1-3]/state` which give the current positions of the three arm servo motors. It collates the data from the

motors and publishes messages of type `JointState` to topic `/joint_states`. This is part of the robot model visualisation, the ROS `robot_state_publisher` takes the `/joint_states` topic and calculates the appropriate transforms, published to the `/tf` topic. RViz can then take the transforms, along with the URDF and model the position of the arm as it moves.

Image Publisher

This node interfaces with the Raspberry Pi camera through the Raspicam drivers (not a ROS package) and publishes messages of type `sensor_msgs::Image` to the topic `/camera_image`.

The node takes settings defined in the launch file from the parameters server which define image resolution, and whether video stabilisation is turned on. These settings are defined dynamically to allow optimisations based on network speed.

In addition, the `republish` node in the ROS `image_transport` package is then used, subscribing to the `/camera_image` node, then compressing the image out to `/camera_repub`. This significantly increases the framerate on the robot driver's viewing computer.

Joystick Converter

This node is the interface between the bare joystick drivers and the desired actions. It subscribes to the given topics of type `sensor_msgs::Joy` and publishes movement vectors of type `geometry_msgs::TwistStamped` to the `/motor_command_vector` and `/servo_command_vector` topics.

Default reset positions are defined in the launch file and taken from the parameter server at startup. The purpose of these is to give an fast way to reset the arm and camera position in the case that the robot driver is confused. These are set to the A, B, X, Y buttons on the controller.

The L1 and R1 buttons are used as dead man's switches to to prevent accidental input. L1 is required to be held down for wheel movement, and R1 is needed for camera and arm movement.

URDF

Results

In our project demonstration our Emu's performed well, successfully navigating the rescue course and identifying the victims. The robot operator was capable of

controlling the robot using the combination of the camera stream and the robot position model. The narrow angle of the camera, and weight imbalances were the primary difficulties.

Future research

Possible additions to this project should increase the ease of naviagtion from the perspecive of the external robot viewer. Currently the camera vision is limited by the network bandwith, which requires us to use a small resolution of 320 * 240 and compressed image quality to acheive high frame rates. The use of the 5GHz network in the Runswift lab would significantly improve this.

The addition of a gyroscope to read the angle the robot is tilted would greatly aid the operator in knowing the position and can be easily incorporated into the 3D URDF model to be viewed in RViz.

References