

Autonomous Ground Support Equipment Design, Construction & Implementation NASA Student Launch Initiative 2014-2015

Pranav Srinivas Kumar & William Emfinger
Vanderbilt Aerospace Club

The 2014-15 Aerospace Club was sponsored by the
Department of Mechanical Engineering and the Boeing
Corporation.

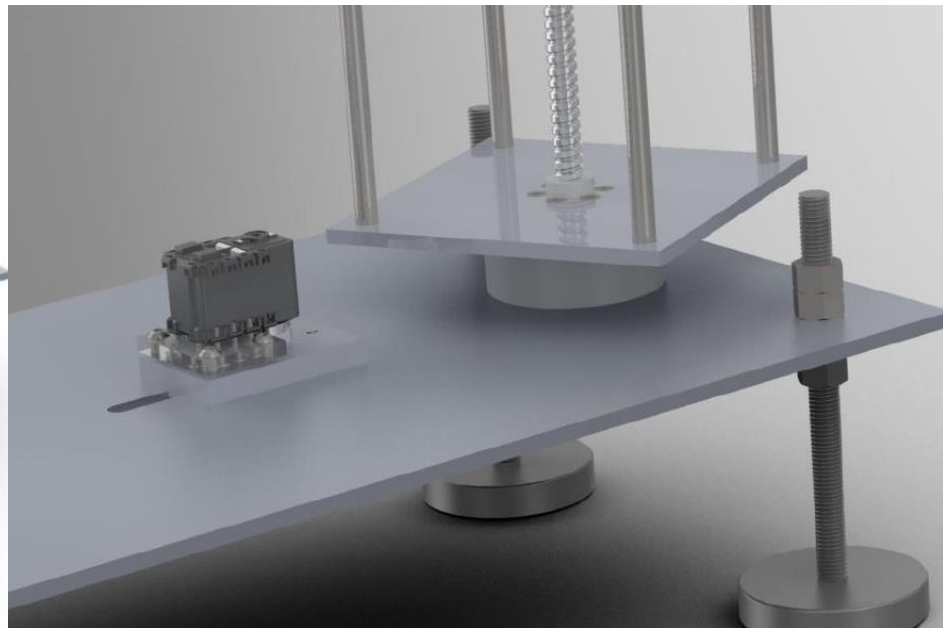
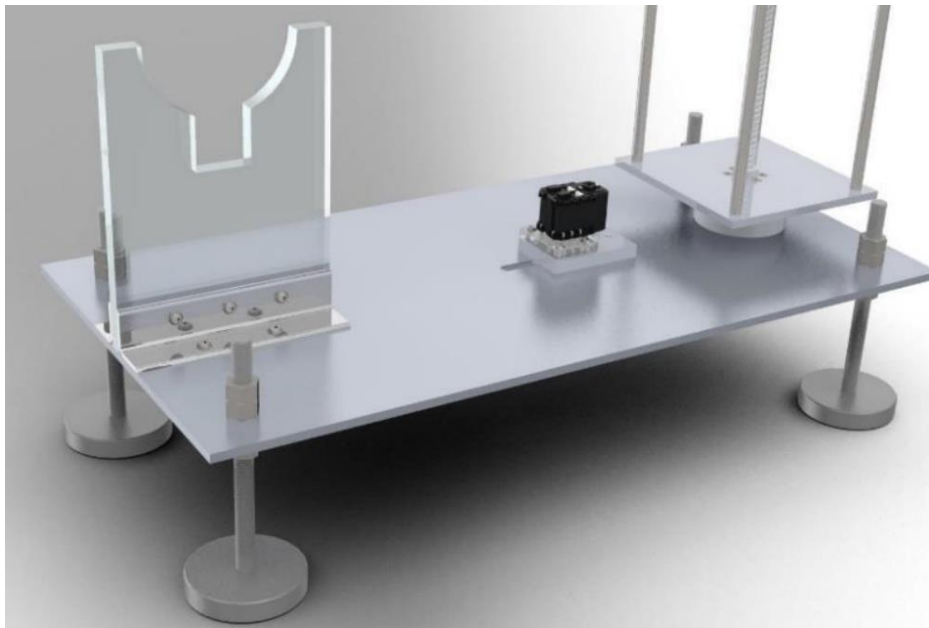
Overview

- Autonomous Ground Support
 - Sample Retrieval
- Design Goals
 - Reliability
 - Simplicity
 - Robustness
- Design Considerations
 - Minimum degrees of freedom
 - Image Processing
 - Distributed Embedded System
 - User Input Panel
 - Model-driven software development



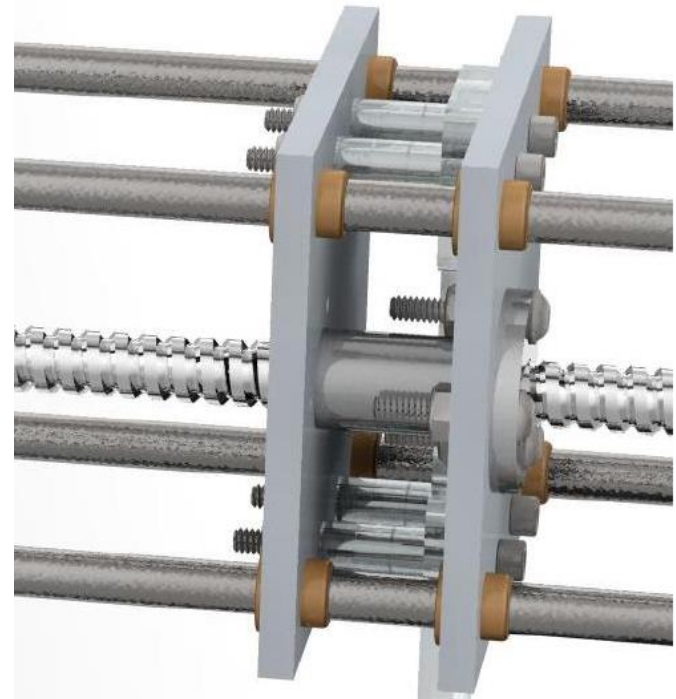
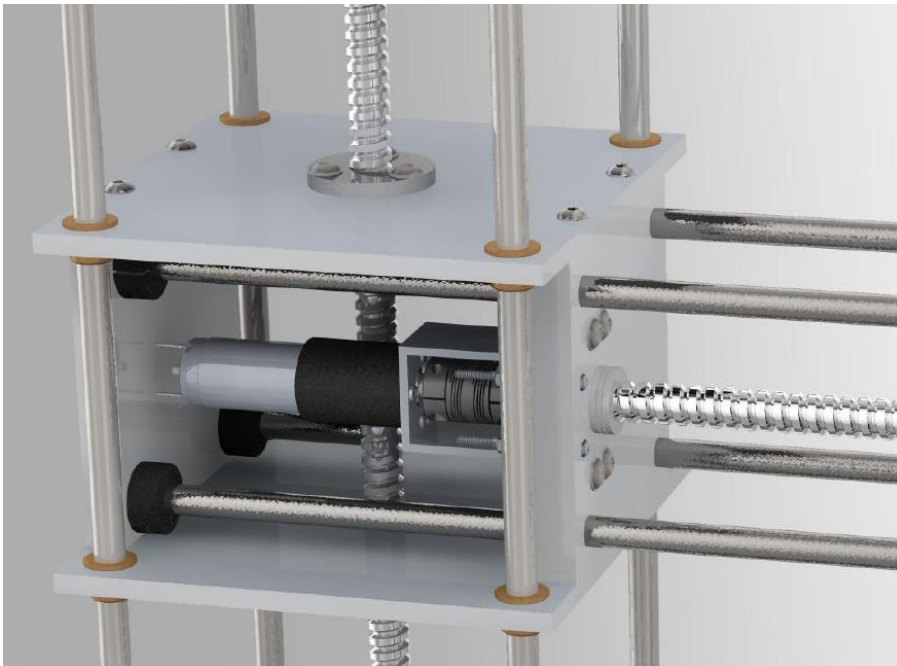
Mechanical Construction (1/3)

- AGSE Base Table
 - 25.5" tall and up to 19.5" radial reach
 - Adjustable leveling feet
 - Rocket stand and Base rotation system



Mechanical Construction (2/3)

- Lead Screw Linear Actuation
 - Faulhaber 12V, 8.1W motors
 - Vertical carriage supports a horizontal powertrain

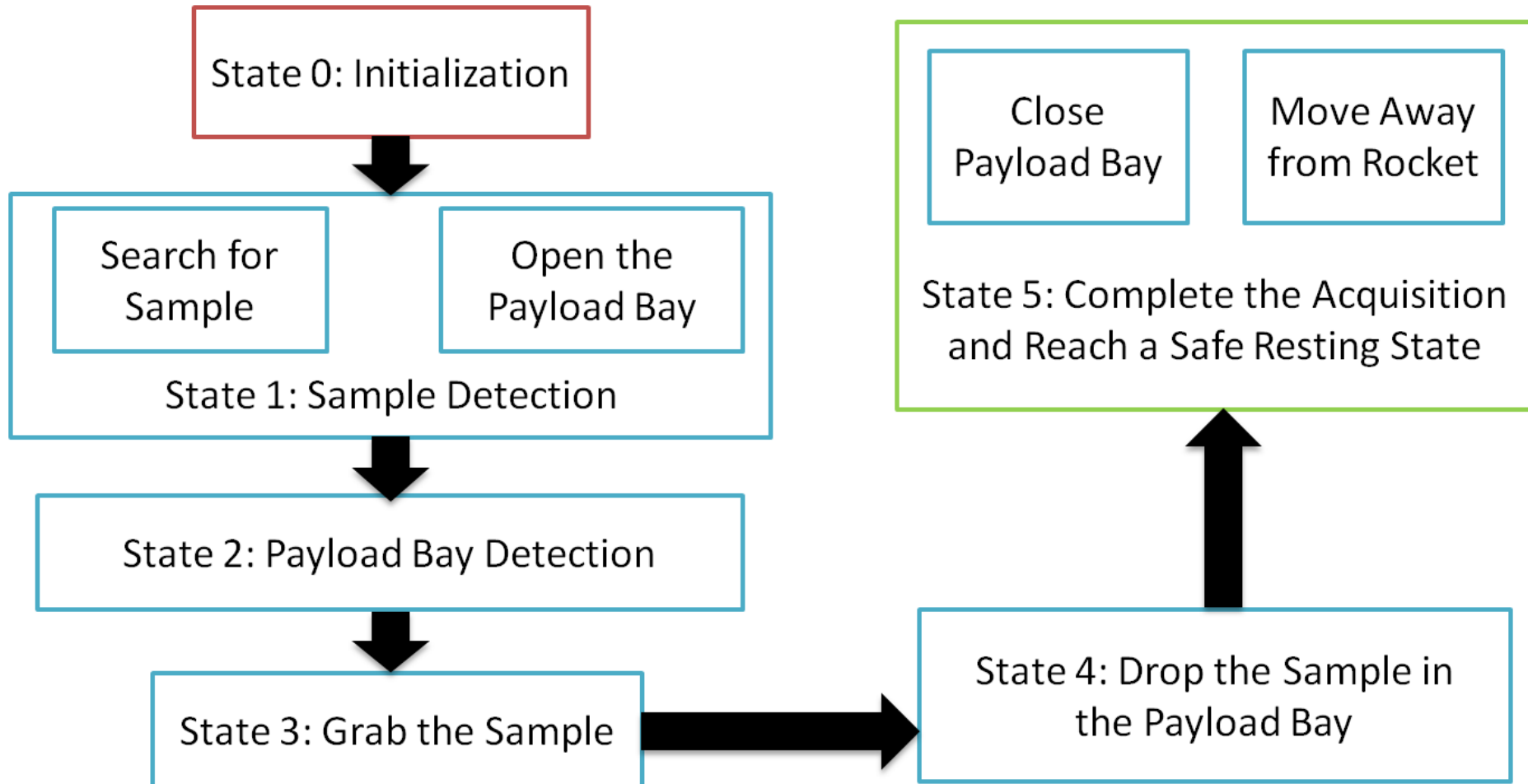


Mechanical Construction (3/3)

- AGSE Gripper
 - Optimized phalange curvature
 - Acrylic “gear box”



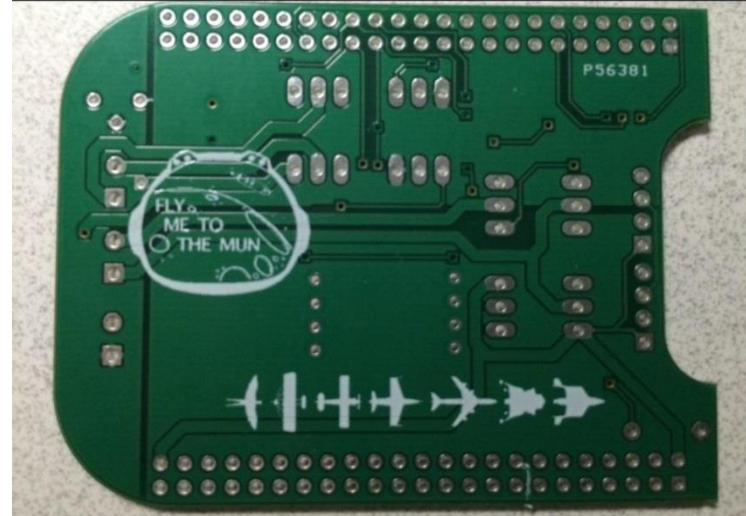
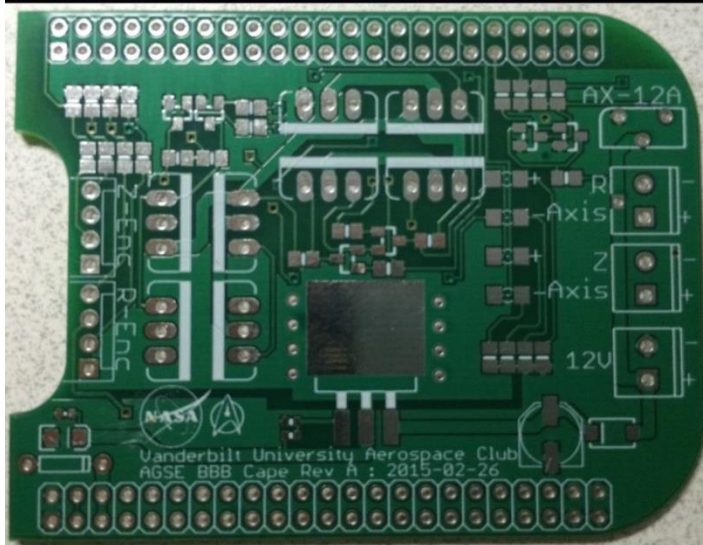
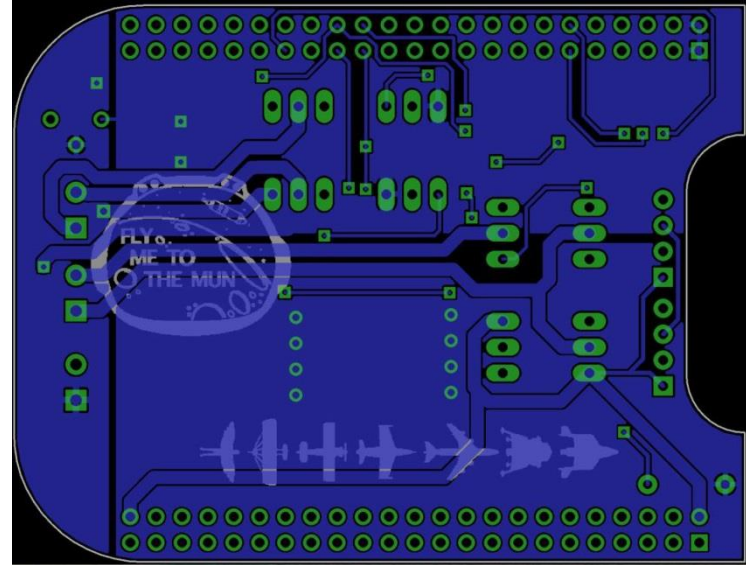
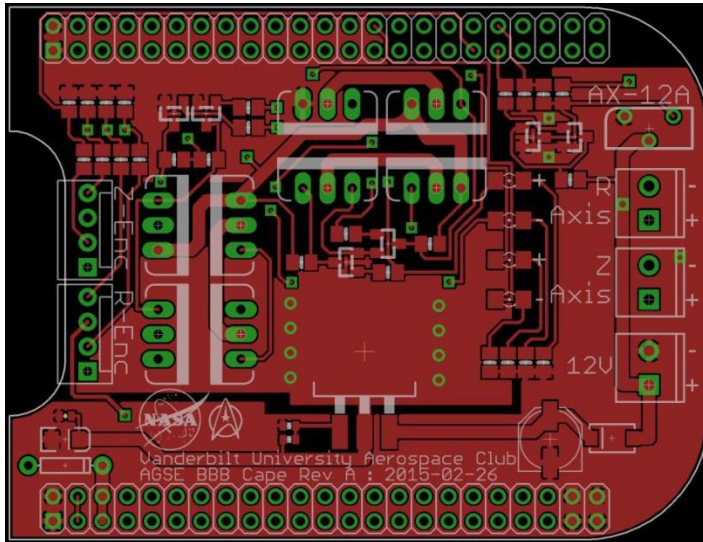
AGSE Overview



AGSE Electronics (1/3)

- Custom PCB
 - Attaches to the BeagleBone Black as a cape
- Provides power & functionality for
 - Serial buffer for UART
 - 3.3V to 5V for servo control
 - H-Bridge for Vertical Actuation Motor
 - H-Bridge for Radial Actuation Motor
 - Radial axis quadrature encoder
 - Vertical axis quadrature encoder

AGSE Electronics (2/3)



AGSE Electronics (3/3)

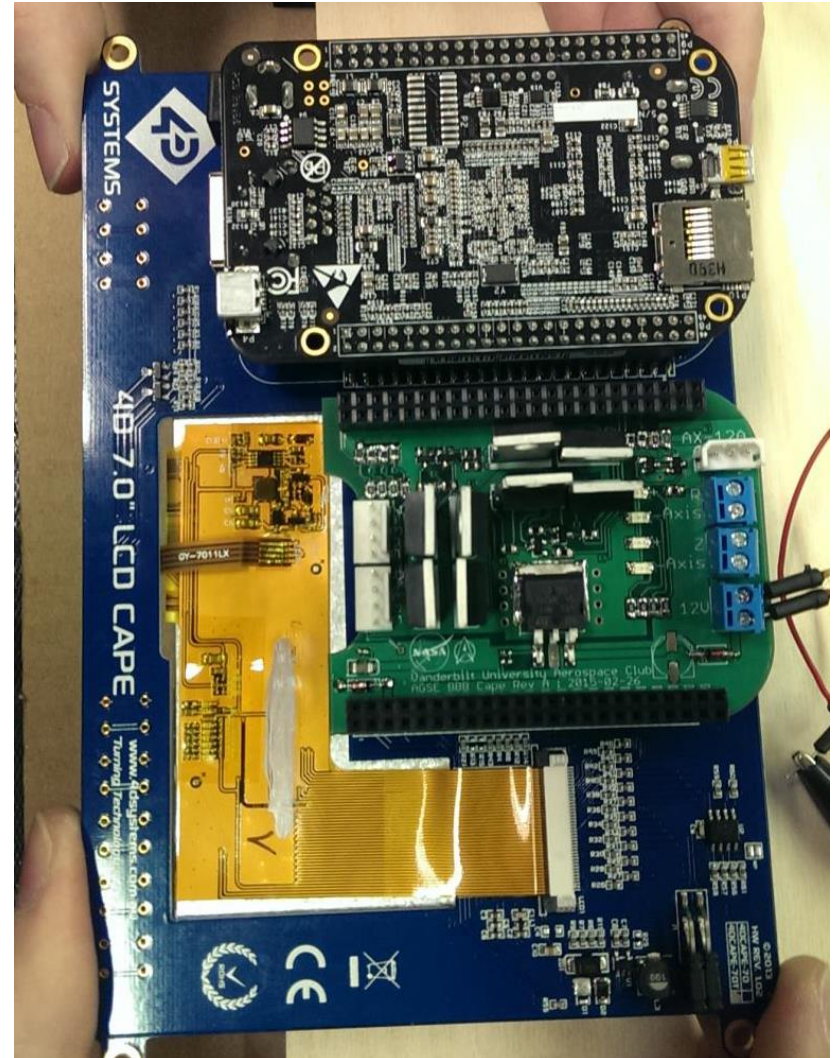
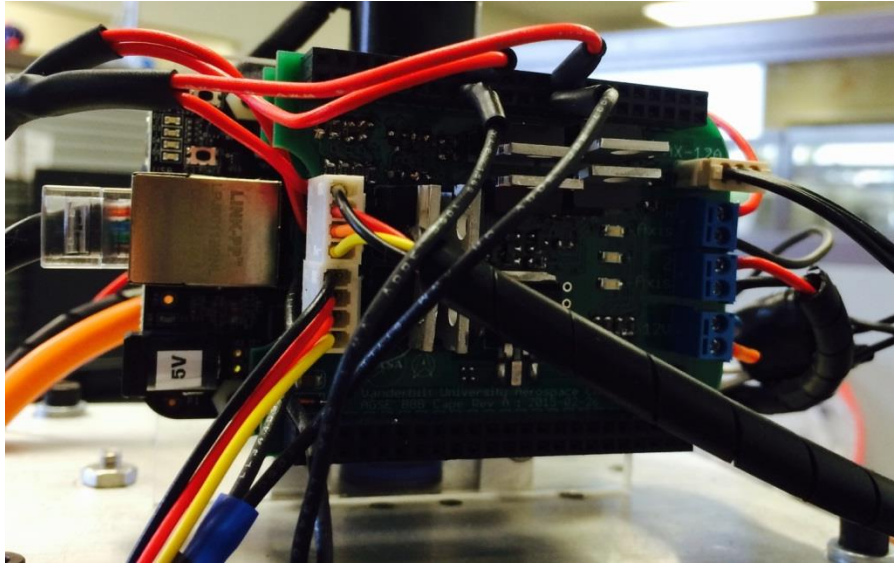


Image Processing – Sample Detection

- OpenCV software library
- Computer vision to classify and track observed objects in real-time
- Periodic Processing
 - Convert RGB image frame to HSV/Grayscale image frame
 - Apply thresholding, filtering, erosion and dilation procedures
 - Draw contours around detected objects
 - Calculate position & relative orientation

Real-time Object Tracking



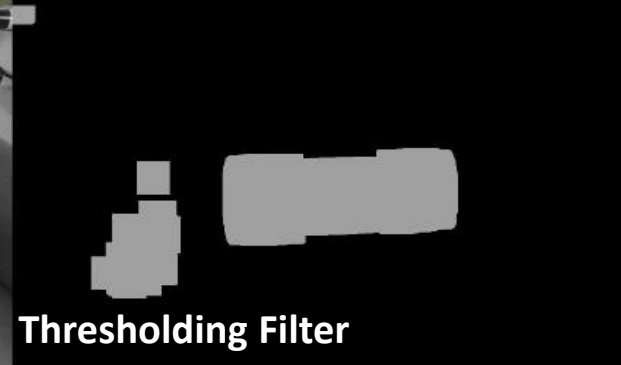
Webcam



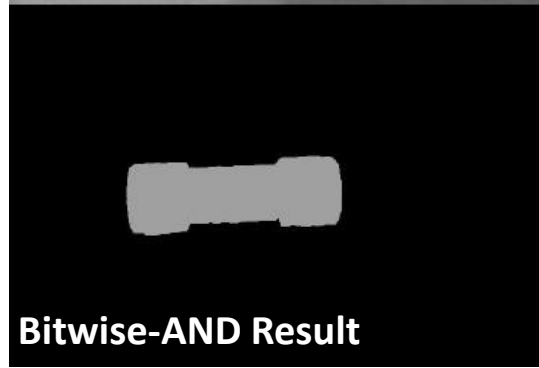
HSV Filter



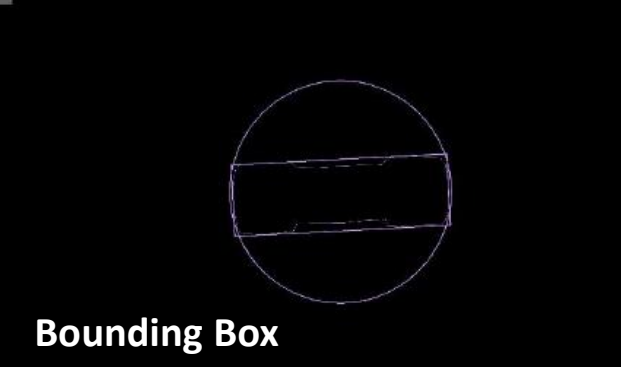
Grayscale



Thresholding Filter



Bitwise-AND Result



Bounding Box

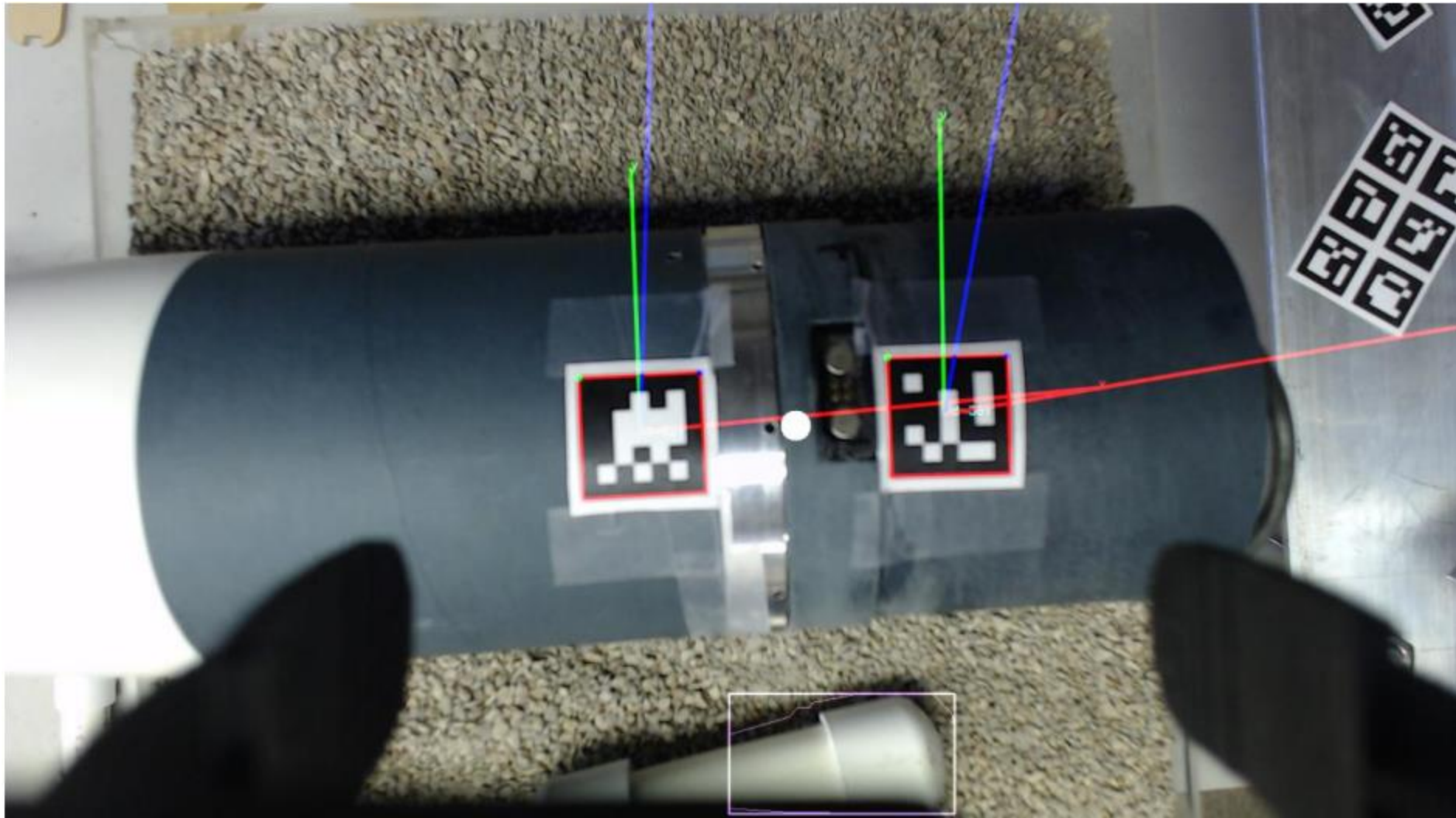
Feature-based Filtering Heuristics



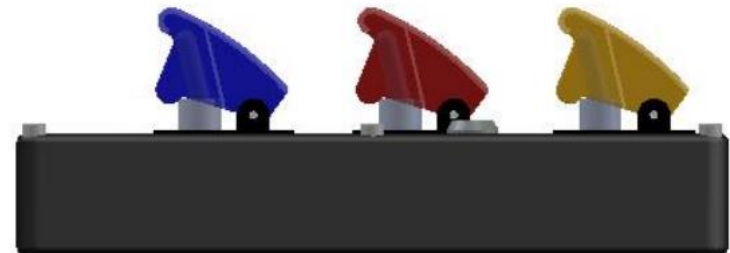
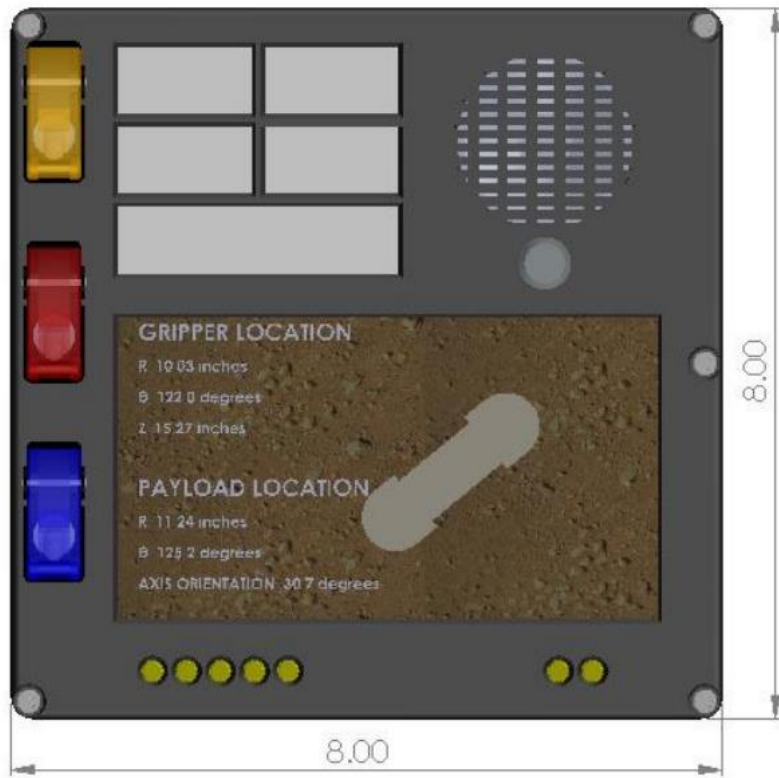
Multiple Sample Detection



Payload Bay Detection

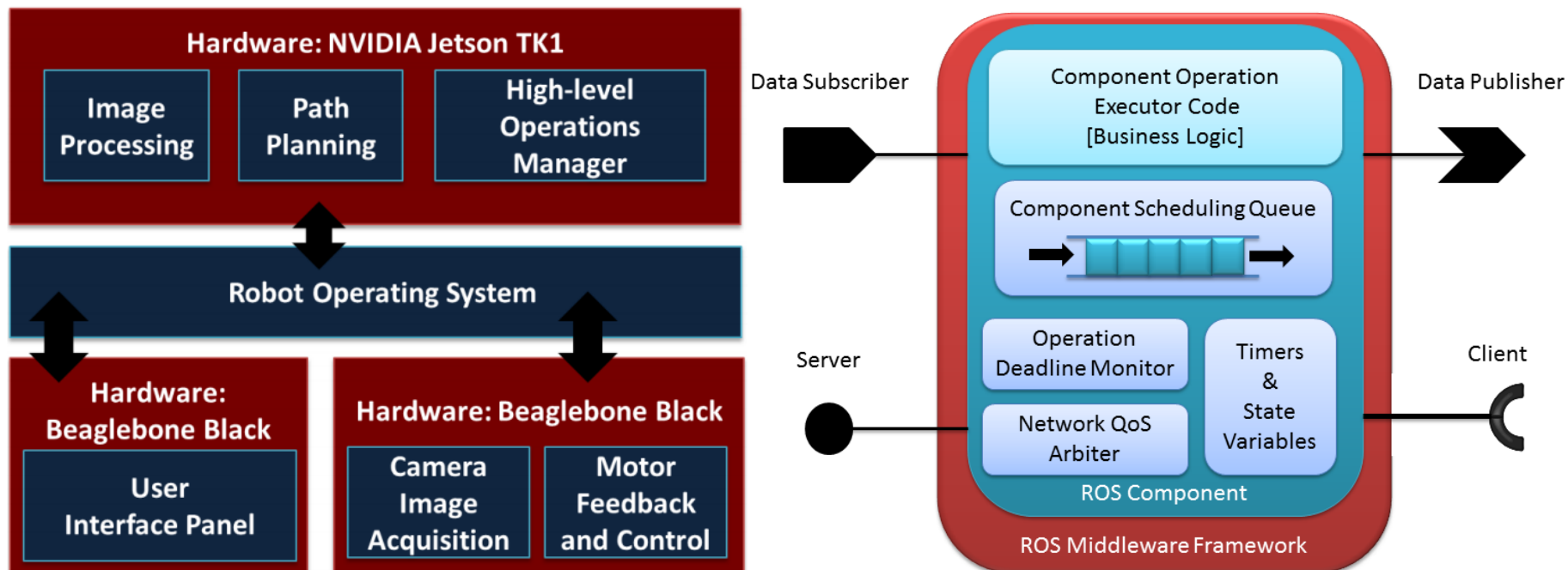


User Input Panel



Software Development Robot Operating System & ROSMOD

- Distributed Embedded System
- Use Robot Operating System
 - ROSMOD Component Model
 - Modeling, Code Generation & Deployment tool suite



ROSMOD Projects

- Software Workspace Model
 - Messages, Services & Component definitions
 - *Workspace & build system generation*
- Hardware Model
 - IP Address, Architecture & network topology
- Deployment Model
 - Process-to-Hardware mapping
 - Component Instantiation in processes
 - *Deployment-specific XML Generation*

Textual Language

```
/*
 * ROSMOD Software Model
 */

// ROSMOD Package - motor_control
package motor_control
{
    // ROSMOD Component - radial_actuator_controller
    component radial_actuator_controller : Base
    {
        // ROSMOD Server - radialPos_server
        server <motor_control/radialPos> radialPos_server
        {
            priority = 50;
            deadline = 0.1;
        }
        // ROSMOD Subscriber - controlInputs_sub
        subscriber <high_level_control/controlInputs> controlInputs_sub
        {
            priority = 50;
            deadline = 0.1;
        }
        // ROSMOD Timer - radialPosTimer
        timer radialPosTimer
        {
            period = 0.01;
            priority = 50;
            deadline = 0.01;
        }
    }
    // ROSMOD Component - vertical_actuator_controller
    component vertical_actuator_controller : Base
    {
        // ROSMOD Server - verticalPos_server
        server <motor_control/verticalPos> verticalPos_server
        {
            priority = 50;
            deadline = 0.1;
        }
        // ROSMOD Subscriber - controlInputs_sub
        subscriber <high_level_control/controlInputs> controlInputs_sub
        {
```

```
/*
 * ROSMOD Hardware Model
 */

// Hardware - BBB_motor_controller
hardware BBB_motor_controller
{
    ip_address = "10.1.1.2";
    username = "debian";
    sshkey = "~/ssh/id_rsa_jetsontk1";
    deployment_path = "/home/debian/";
    arch = armv7l;
}
// Hardware - BBB_user_input
hardware BBB_user_input
{
    ip_address = "10.1.1.3";
    username = "debian";
    sshkey = "~/ssh/id_rsa_jetsontk1";
    deployment_path = "/home/debian/";
    arch = armv7l;
}
// Hardware - Jetson_TK1
hardware Jetson_TK1
-:--- agse.rhw      Top L8      Git-master (Antlr.Java/l)
/*
 * ROSMOD Deployment Model
 */

// ROSMOD Hardware Model - agse
using agse;

// ROSMOD Node - arm
node arm
{
    properties
    {
        {
            ref = "agse/Jetson_TK1";
            priority = 50;
        }
    }
    component_instance arm_controller_i
    {
```

ROSMOD Editor – Software Model

The screenshot displays the ROSMOD Editor interface. At the top, the title bar reads "ROSMOD Editor". Below it is a toolbar with icons for file operations and a "Software:" dropdown menu. The main workspace shows a software model diagram with the following components:

- motor_control** package containing:
 - gripperRotation** (SRV)
 - verticalPos** (SRV)
 - radialPos** (SRV)
- radial_actuator_controller** package containing:
 - radialPos_server** (SERVER)
 - controlInputs_sub** (SUB)
 - radialPosTimer** (TIMER)
- vertical_actuator_controller** package containing:
 - verticalPos_server** (SERVER)

An "Edit Service" dialog is open on the left, showing the definition for the **radialPos** service:

Name: radialPos

Definition:

```
1 int64 goal
2 bool update
3 bool setZeroPosition
4 —
5 int64 current
6 bool lowerLimitReached
7 bool upperLimitReached
8
```

Buttons: Ok, Close

Below the diagram, a status bar indicates: "Editing radialPos of type Service".

At the bottom, the "Console Output" tab is active, showing the following log messages:

```
ROSTOOLS::Opening ROSMOD Project: /home/kelsier/Repositories/agse2015/code/agse_rosmod_v3
ROSTOOLS::Project Name: agse_rosmod_v3
ROSTOOLS::Project Path: /home/kelsier/Repositories/agse2015/code/agse_rosmod_v3
ROSTOOLS::Parsing Software files
ROSTOOLS::Parsing Message files
ROSTOOLS::Parsing Service files
ROSTOOLS::Parsing Abstract Business Logic files
ROSTOOLS::Parsing Port Network Profile files
```

ROSMOD Editor – Hardware Model

The screenshot displays the ROSMOD Editor interface for editing hardware models. The main workspace has a yellow background and contains two hardware models: 'BBB_motor controller' and 'Jetson TK1'. The 'BBB_motor controller' model is highlighted with a red box. An 'Edit Hardware' dialog box is open, showing the following fields:

- Name: BBB_motor_controller
- IP Address: 10.1.1.2
- Username: debian
- SSH Key: ~/.ssh/id_rsa_jetsontk1
- Deployment Path: /home/debian/ (highlighted with an orange border)
- Initialization Script: (empty)
- Architecture: armv7l

The 'Console Output' tab at the bottom shows the following logs:

```
ROSTOOLS::Opening ROSMOD Project: /home/kelsier/Repositories/agse2015/code/agse_rosmod_v3
ROSTOOLS::Project Name: agse_rosmod_v3
ROSTOOLS::Project Path: /home/kelsier/Repositories/agse2015/code/agse_rosmod_v3
ROSTOOLS::Parsing Software files
ROSTOOLS::Parsing Message files
ROSTOOLS::Parsing Service files
ROSTOOLS::Parsing Abstract Business Logic files
ROSTOOLS::Parsing Port Network Profile files
```


ROSMOD Editor – Deployment Model

The screenshot displays the ROSMOD Editor interface for a project named 'agse'. The main workspace shows a deployment model with several nodes represented by blue squares with labels. The nodes are organized into groups: 'arm' (containing 'arm_controller_i'), 'detector' (containing 'image_processor_i'), 'positioning' (containing 'vertical_controller_i'), 'imager' (containing 'image_sensor_i'), and 'ser' (containing 'ser'). The 'arm' and 'image_processor_i' nodes are highlighted with red rectangles. An 'Edit Node' dialog box is open, showing the configuration for the 'arm' node. The dialog includes fields for Name (arm), Hardware (Jetson_TK1), Priority (50), and Command-line Arguments. The 'Ok' and 'Close' buttons are visible at the bottom of the dialog. Below the workspace, the 'Console Output' tab is active, showing the following log messages:

```
ROSTOOLS::Opening ROSMOD Project: /home/kelsier/Repositories/agse2015/code/agse_rosmod_v3
ROSTOOLS::Project Name: agse_rosmod_v3
ROSTOOLS::Project Path: /home/kelsier/Repositories/agse2015/code/agse_rosmod_v3
ROSTOOLS::Parsing Software files
ROSTOOLS::Parsing Message files
ROSTOOLS::Parsing Service files
ROSTOOLS::Parsing Abstract Business Logic files
ROSTOOLS::Parsing Port Network Profile files
```

Workspace Code Generation

image_processor

```
SERVICE sampleStateFromImage_server
SERVICE payloadBayStateFromImage_server
CLIENT captureImage_client
PUB sampleDetectionImages_pub
PUB payloadBayDetectionImages_pub
SUB controlInputs_sub
```

image_sensor

```
SERVICE captureImage_server
SUB controlInputs_sub
```

user_input_imager

```
CLIENT captureImage_client
```

```
src
├── image_processing
│   ├── CMakeLists.txt
│   └── include
│       └── image_processing
│           ├── Component.hpp
│           ├── image_processor.hpp
│           ├── image_sensor.hpp
│           ├── Logger.hpp
│           ├── rapidxml.hpp
│           ├── rapidxml_utils.hpp
│           ├── user_input_imager.hpp
│           └── xmlParser.hpp
├── msg
│   ├── payloadBayDetectionImages.msg
│   └── sampleDetectionImages.msg
├── package.xml
├── src
│   └── image_processing
│       ├── Component.cpp
│       ├── image_processor.cpp
│       ├── image_sensor.cpp
│       ├── Logger.cpp
│       └── user_input_imager.cpp
└── srv
    ├── captureImage.srv
    ├── payloadBayStateFromImage.srv
    └── sampleStateFromImage.srv
```

Generated Skeleton Code

```
class image_processor : public Component
{
public:
    // Constructor
    image_processor(ComponentConfig& config, int argc, char **argv) : Component(config, argc, argv) {}

    // Initialization
    void Init(const ros::TimerEvent& event);

    // Subscriber Callback - controlInputs_sub
    void controlInputs_sub_OnOneData(const high_level_control::controlInputs::ConstPtr& received_data);

    // Server Callback - sampleStateFromImage_server
    bool sampleStateFromImageCallback(image_processing::sampleStateFromImage::Request &req,
        image_processing::sampleStateFromImage::Response &res);

    // Server Callback - payloadBayStateFromImage_server
    bool payloadBayStateFromImageCallback(image_processing::payloadBayStateFromImage::Request &req,
        image_processing::payloadBayStateFromImage::Response &res);

    // Start up
    void startUp();

    // Destructor
    ~image_processor();

private:
    // Subscriber
    ros::Subscriber controlInputs_sub;

    // Publisher
    ros::Publisher sampleDetectionImages_pub;

    // Publisher
    ros::Publisher payloadBayDetectionImages_pub;

    // Server
    ros::ServiceServer sampleStateFromImage_server;

    // Server
    ros::ServiceServer payloadBayStateFromImage_server;

    // Client
    ros::ServiceClient captureImage_client;

    /// Start User Globals Marker
    /// End User Globals Marker

    // Initialization Function
    /// Start Init Marker
    void image_processor::Init(const ros::TimerEvent& event)
    {
        // Initialize Here

        // Stop Init Timer
        initOneShotTimer.stop();
    }
    /// End Init Marker

    // Subscriber Callback - controlInputs_sub
    /// Start controlInputs_sub_OnOneData Marker
    void image_processor::controlInputs_sub_OnOneData(const high_level_control::controlInputs::ConstPtr& received_data)
    {
        // Business Logic for controlInputs_sub Subscriber
    }
    /// End controlInputs_sub_OnOneData Marker

    // Server Callback - sampleStateFromImage_server
    /// Start sampleStateFromImageCallback Marker
    bool image_processor::sampleStateFromImageCallback(image_processing::sampleStateFromImage::Request &req,
        image_processing::sampleStateFromImage::Response &res)
    {
        // Business Logic for sampleStateFromImage_server Server
        return true;
    }
    /// End sampleStateFromImageCallback Marker

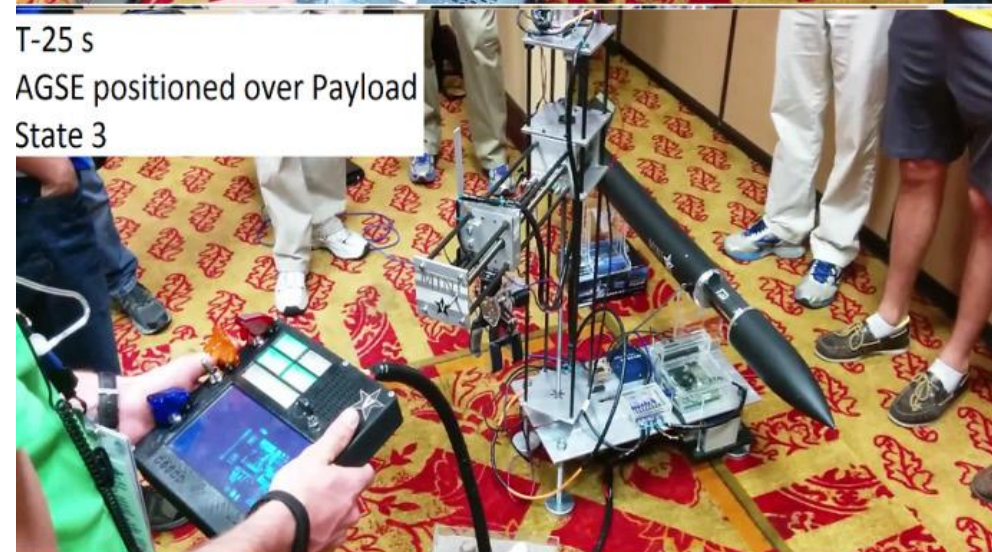
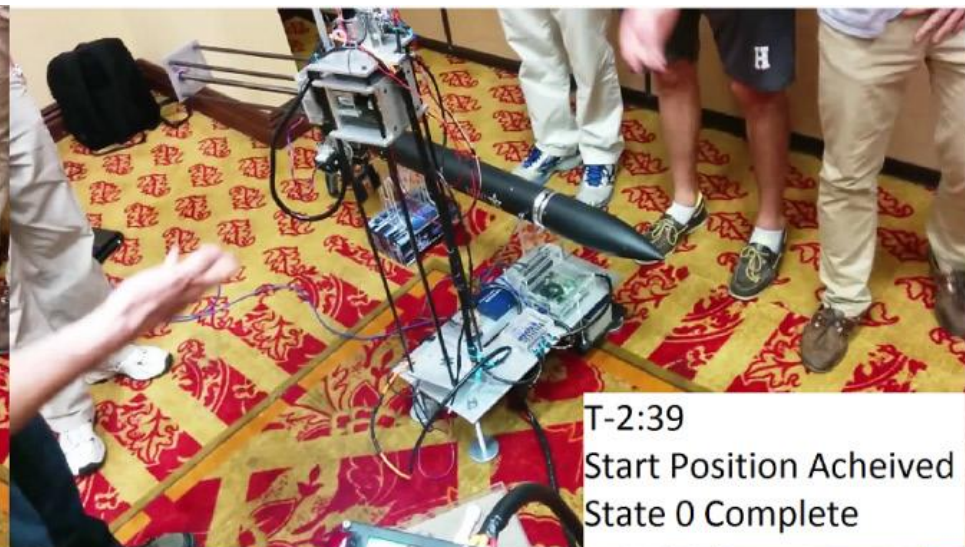
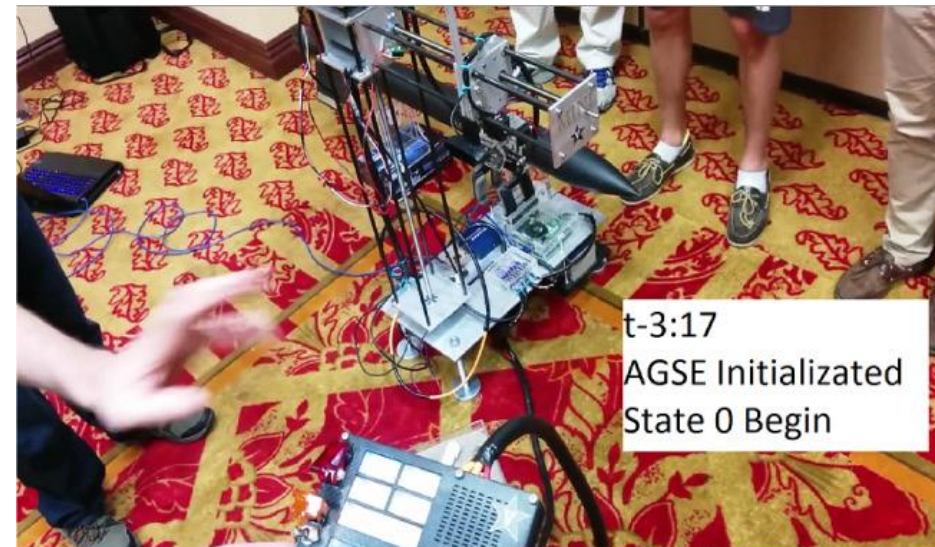
    // Server Callback - payloadBayStateFromImage_server
    /// Start payloadBayStateFromImageCallback Marker
    bool image_processor::payloadBayStateFromImageCallback(image_processing::payloadBayStateFromImage::Request &req,
        image_processing::payloadBayStateFromImage::Response &res)
    {
        // Business Logic for payloadBayStateFromImage_server Server
        return true;
    }
    /// End payloadBayStateFromImageCallback Marker

    // Destructor - Cleanup Ports & Timers
};
```

----- image_processor.hpp 27% L26 (C++/l Abbrev)
Beginning of buffer

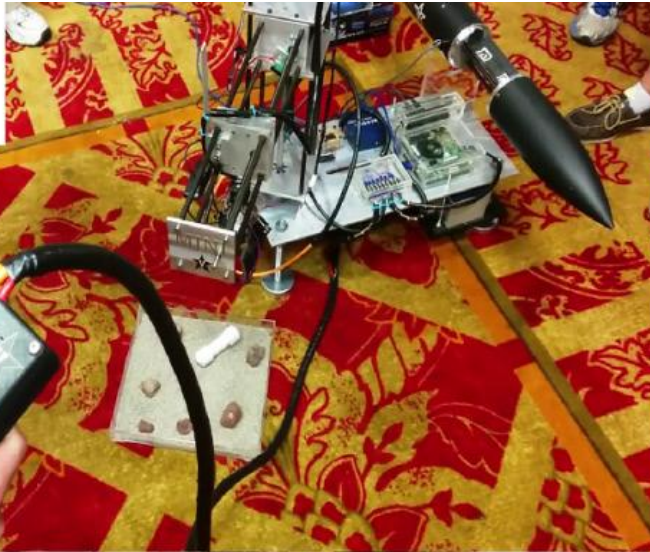
```
----- image_processor.cpp Top L44 (C++/l Abbrev)
```


Performance Assessment



Performance Assessment

T-1:18
Payload Found
State 1 Complete



T-32 s
Payload Bay Found
State 2 Complete



T+28 s
Place sample in
Payload Bay
State 4 Complete



T+57 s
Payload Bay Closed
State 5 Complete

Summary

- Model-driven Component-based Engineering
- Distributed Managed Embedded System
- Robot Operating System & ROSMOD
- Autonomous Sample Retrieval
 - Periodic Image Processing
 - Unknown Sample & Rocket location
 - Under 5 minutes on Competition Day
- We won 😊

Vanderbilt Aerospace Club 2014-2015

