Vision based autonomous robotic control for advanced inspection and repair

Walter S. Wehner Jr.
KENNEDY SPACE CENTER
Major: M.S. Computer Science

KSC FO Spring Session
Date: 09 04 2014

Vision based autonomous robotic control for advanced inspection and repair

Walter S. Wehner[1]
New Jersey Institute of Technology, Newark, NJ, 07103

The advanced inspection system is an autonomous control and analysis system that improves the inspection and remediation operations for ground and surface systems. It uses optical imaging technology with intelligent computer vision algorithms to analyze physical features of the real-world environment to make decisions and learn from experience. The advanced inspection system plans to control a robotic manipulator arm, an unmanned ground vehicle and cameras remotely, automatically and autonomously. There are many computer vision, image processing and machine learning techniques available as open source for using vision as a sensory feedback in decision-making and autonomous robotic movement. My responsibilities for the advanced inspection system are to create a software architecture that integrates and provides a framework for all the different subsystem components; identify open-source algorithms and techniques; and integrate robot hardware.

Nomenclature

AIS	=	advanced inspection system
HMI	=	human machine interface
UGV	=	unmanned ground vehicle
VMSS	=	video management server system
IHM	=	integrated health management
ROS	=	robot operating system
OS	=	operating system
API	=	application programing interface
RPC	=	remote procedure call
VMS	=	video management software
SDK	=	software development kit
GUI	=	graphical user interface
AIROS	=	advanced inspection and repair operating system
SDF	=	simulation description format
USRA	=	Universities Space Research Association
KSC	=	Kennedy Space Center

I. Introduction

The Advanced Inspection System (AIS) is a project that will investigate and develop hardware and software to automate the inspection and repair tasks for ground systems such as cryogenic fueling systems. AIS will accomplish this by automating data collection, image processing, archiving, decision-making, and the execution of inspection and repair activities. AIS will develop intelligent computer vision algorithms to analyze physical features of the real-world environment to produce decision-making information for autonomous planning and control of an unmanned ground vehicle and a robotic manipulator arm. Computer vision, machine vision, image processing, artificial intelligence, machine learning and robotics are some of the many different fields in computer science that AIS will investigate to accomplish autonomous control. Writing and applying computer vision algorithms for performing autonomous robotic control is a challenging effort but before an algorithm can be written a framework

[1] KSC NASA Intern, NE, Kennedy Space Center, New Jersey Institute of Technology.

for communication and interconnectivity is needed. Defining a software architecture that is flexible, reliable and extensible supporting an autonomous control system will require integrating aspects from these fields of computer science to support real-time processing of sensor data for making enhanced decisions and autonomously controlling the entire system.

My project is to develop a robust, innovative, efficient and flexible software architecture that is extensible and capable of achieving the goal of autonomous operations. AIS will use many different software technologies over varying hardware platforms including sensors, cameras and robots. The major components of the AIS are a high-speed wired and wireless data network, a distributed and autonomic computing system, visible and spectral band cameras, contact and non-contact sensors, unmanned vehicles, robotic manipulators, highly-interactive human-machine-interfaces (HMIs), vision and artificial intelligence algorithms, and reliable high-capacity data storage. It is my goal in designing the software architecture to create a common interface for integrating all the different components to facilitate autonomous field inspections, repairs, decision-making, automated data collection, and image processing and analysis.

The software architecture will be the framework for the development of the computer vision and system intelligence algorithms. The AIS autonomous control and decision software will need to be tested before it is deployed on physical hardware. Using simulation software to model the hardware and environment will allow for refinement of the autonomous software systems before deployment on actual hardware. The unmanned ground vehicle (UGV) is currently modeled in the physics simulator, but the robotic manipulator that is attached to the UGV is not. I will begin the project of adding the robotic manipulator arm into the model for use in the physics simulation.

II. Software Architecture

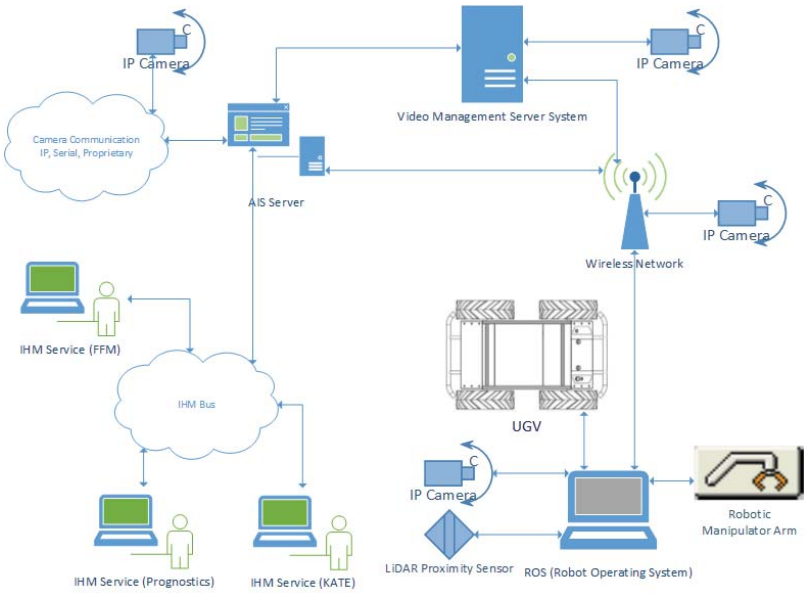

Figure 1. AIS System Architecture.

The software architecture for an autonomous robotic system requires asynchronous fault tolerant communications, distributed parallel processing and extensibility. To accomplish these goals, AIS is divided into three subsystems the UGV, the Video Management Server System (VMSS) and the AIS Server. The UGV subsystem is our autonomous robot equipped with multiple sensors, a robotic manipulator arm and a camera. The VMSS performs all video capture and distribution. The AIS Server provides system intelligence and interfaces to the rest of the Integrated Health Management (IHM) services.

The three subsystems are interdependent on each other for sensor and decision data and control messaging. A challenge to the software architecture is that each subsystem has a different software environment. AIS will develop a common communications framework that each subsystem will use to address this challenge. Having a common communications framework will create a layer of abstractions between the different subsystem software environments and provide flexibility for future expansion.

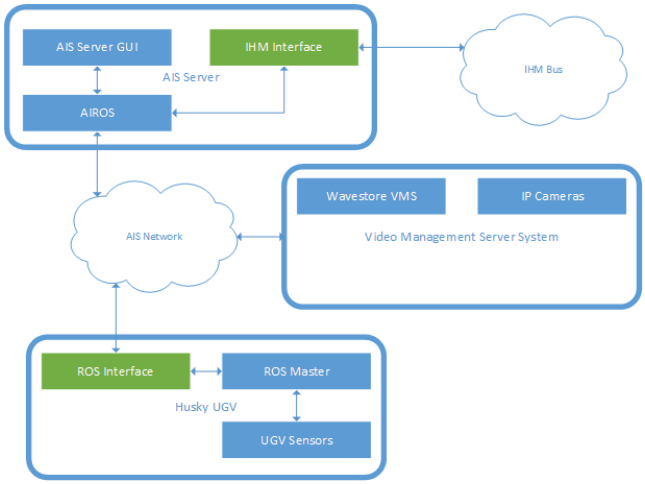

Figure 2. AIS Software Architecture.

A. Unmanned Ground Vehicle (UGV)

The UGV uses robot operating system (ROS), which is a layer on top of Ubuntu Linux. ROS provides tools, libraries, and services for robot interaction, control, autonomous navigation, and sensor monitoring. ROS itself is not a full operating system, but it is commonly referred to as a meta-operating system. Robotics requires high-level features such as concurrency, inter-communication and extensibility. These tasks are difficult to implement using only an operating systems (OS) raw application programing interface (API) but ROS extends the traditional OS and provides hierarchical abstraction and management of running programs; communication between programs; and a collection of powerful programs and code libraries as extension.

ROS provides a communications platform that follows a publish/subscribe paradigm. The basic units of ROS are nodes, topics and services. Nodes publish to topics and nodes subscribe to topics. Nodes provide services and access services of other nodes. A topic is like a mailbox holding information to be retrieved by nodes and receiving new information from nodes. A service provides an implementation of a function that can return a result to the calling node. Parameters can be passed to the service, have the service perform a function, and return a result to the calling node. When nodes and topics one way communications are not appropriate, services provide remote procedure call (RPC) request/reply functionality.

B. Video Management Server System (VMSS)

The VMSS is a Linux server that runs a proprietary video management software (VMS) application to manage, distribute, and store imagery data. This software allows users to securely access the VMSS and the data stored on it from a client application. The server includes a scripting language based on LUA script, that allows actions to be scripted and triggered based on events within the server.

Although the VMS is closed source there is a Windows-based ActiveX Software Development Kit (SDK). The SDK enables software developers to interface and access certain features of the server-based VMS for custom application integration. Since the VMS application is proprietary, the AIS project will use the SDK on the AIS Server to extend certain functions of the VMS and develop custom AIS computer vision algorithms.

C. Advanced Inspection System (AIS) Server

The AIS Server provides three components: a graphical user interface (GUI), the advanced inspection and repair operating system (AIROS) and the IHM interface for communication between AIS and external IHM services. The GUI will be implemented as a web application and provide the maintenance and administration portal for all AIS subsystems. AIROS is a custom operating system that will handle communication between all AIS subsystems, communication to the IHM interface and provide system intelligence. The IHM Interface is the component that will serve as a communications proxy between AIS and IHM converting messages between formats.

The AIS Server GUI is written in ASP.NET/C# and will communicate to AIROS over the AIROOS communications layer. The GUI will provide an administration interface for all the AIS subsystems. There are many benefits to this design including portability between platforms; flexibility in GUI design and application

implementation; and reusability of image processing and computer vision functions. The GUI is also used to administer the AIROS Core services available to the IHM Interface.

AIROS is composed of two application services the AIROS Core and the AIROS Interface. The AIROS Core software is composed of four libraries that provide the system functionality and intelligent algorithms. The AIROS Interfaces are composed of two libraries that provide a common interface for external components to communicate with AIROS.

The IHM Interface component will allow messages to be received and published by the AIROS Core to/from other IHM services. The IHM interface will take IHM messages and create AIROS message packages. Having an abstraction between the IHM message structure and the AIROS message structure is advantageous because when IHM messages change, AIROS will not be impacted.

III. Integrating Robotics Hardware

Creating a sustainable software architecture is one aspect of AIS, robotic hardware integration is another. To take advantage of the software architecture AIS will integrate all the custom robotics hardware into ROS. Integrating hardware requires many steps such as creating the proper model files, creating the proper ROS nodes and topics and creating software plugins to simulate hardware for verification and testing.

A. Robot Model

A robot model gives the physical description of the different pieces of a robot. Each segment and joint of a robot is described and properties such as mass, inertia, static and dynamic friction, collision bounds and visual characteristics are defined. There are two different formats for a model file. The model file format used by the simulation software GAZEBO is the Simulation Description Format (SDF).

A robot model file will consist of many link and joint elements. The link element represents a physical link with inertia, collision, and visual properties. The joint element represents the connection between two links with kinematic and dynamic properties.

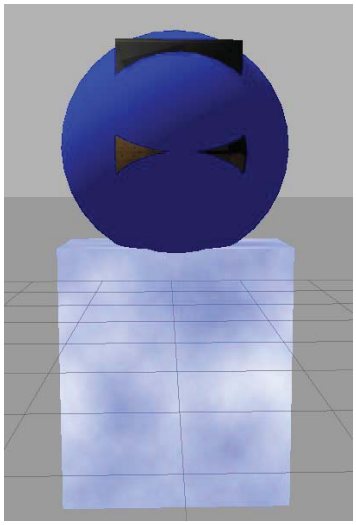

Figure 3. Robot generated from SDF model.

The SDF file is interpreted by GAZEBO and a robot is generated for use in the simulation.

```
<sdf version="1.3">
  <model name='rosbot'>
    <link name="torso">
      <pose>0.1 -0.1 0.4 0 0 0</pose>
      <inertial>
        <mass>1.05</mass>
        <pose>-0.01 0 0.04 0 0 0</pose>
        <inertia></inertia>
      </inertial>
      <collision name="collision">
        <geometry>
          <box>
            <size>.2 .1 .23</size>
          </box>
        </geometry>
      </collision>
      <visual name="visual">
        <pose>0 0 0.0 0 0 3.1</pose>
        <material>
          <script>
            <name>Gazebo/CloudySky</name>
          </script>
        </material>
      </visual>
    </link>
    <joint type="revolute" name="neck_pan">
      <pose>0 0 0.1 0 0 0</pose>
      <child>torso</child>
      <parent>head_pan</parent>
      <axis>
        <limit>
          <lower>-1.57</lower>
          <upper>1.57</upper>
        </limit>
        <xyz>0 0 1</xyz>
      </axis>
    </joint>
    <plugin name="ros_based_plugin"
            filename="libros_model_plugin.so"/>
  </model>
</sdf>
```

Code Block 1. Example SDF model file.

B. GAZEBO Plugin Integration

The physics simulator GAZEBO is a useful tool that allows you to test and verify your ROS node and topic design in a simulation environment. This tool will allow you to implement and test control features in a simulated physics environment before deploying on physical hardware. Creating a plugin allows your GAZEBO models to publish and subscribe to ROS topics. I created a generic set of classes that the AIS development team can use and extend for simulating a three-axis motor.

The motor encoder class holds information about the current location, position and angle, and the time parameter. The motor uses the encoder to simulate movement based on the desired speed or position coordinates that are applied. The motor controller class provides the high level interface to the GAZEBO plugin. The controller takes the different joints and then uses these to create the underlying motors that are used for rotation. The motor controller will accept a speed to apply in a specific rotation axis and then communicates this to the motors. The motors then communicate to the encoders and determine the amount of movement that will occur during the update. This information is given back to the controller that then moves the joint in the GAZEBO simulation.

```
#include <boost/bind.hpp>
#include <gazebo.hh>
#include <physics/physics.hh>
#include <common/common.hh>
#include <stdio.h>
#include "ros/ros.h"
#include "sensor_msgs/Joy.h"

namespace gazebo
{
  class MotorEncoder
  {
    public:
      float Pose, Angle;
      long Clock;

    public: MotorEncoder();
  };

  class Motor
  {
    private:
      MotorEncoder Encoder;

    public:
      float Pose, Angle, Speed, MaxAngle, MaxSpeed, MinAngle, MinSpeed;

    public: Motor();
    public: bool UpdateEncoder(float _pose, float _angle, long _clock);
    public: bool OnUpdate();
  };

  class YawPitchRollMotorController
  {
    public:
      double Torque, YawSpeed, PitchSpeed, RollSpeed;

    private:
      physics::JointPtr yawJoint, pitchJoint, rollJoint;
      Motor YawMotor, PitchMotor, RollMotor;

    public: YawPitchRollMotorController();

    public: bool Load(physics::JointPtr _yawJoint, physics::JointPtr _pitchJoint,
                      physics::JointPtr _rollJoint)
    public: bool Init();
    public: bool OnUpdate();
    public: bool Move(double yawFactor, double pitchFactor, double rollFactor);
  };

  class ROSModelPlugin : public ModelPlugin
  {
    public: ROSModelPlugin() { ros::init(0, NULL, name); }
    public: ~ROSModelPlugin() { delete this->node; }
    public: void Load(physics::ModelPtr _model, sdf::ElementPtr _sdf)
    {
      motorController.Load(_model->GetJoint("neck_pan"),
                           _model->GetJoint("neck_tilt"), NULL);
    }

    public: void Init () { motorController.Init(); }
    public: void OnUpdate() { ros::spinOnce(); }
    void ROSCallback(const sensor_msgs::Joy::ConstPtr& joy)
    {
      motorController.OnUpdate();
      motorController.Move(p_scale*joy->axes[angular_],
                           t_scale*joy->axes[linear_], 0);
    }
    private: YawPitchRollMotorController motorController;
    private: ros::NodeHandle* node;
  };

  GZ_REGISTER_MODEL_PLUGIN(ROSModelPlugin)
}
```

Code Block 2. Example GAZEBO Plugin.

IV. Conclusion

The work presented here is ongoing. AIS will explore this current proposed architecture and begin implementation.

Acknowledgments

I would like to thank Barbara Brown, Jose Perotti and Mark Lewis for selecting me for this great opportunity. I would like to thank the Ground Systems Development and Operations (GSDO) Program, Advanced Ground Systems Maintenance (AGSM) Element Integration Team for providing the funding for my opportunity and the NE-E9 branch for providing me office space. I would like to thank the Universities Space Research Association (USRA), Rose Austin, Benita Desuza, Grace Johnson, Rob Cannon and everyone in the Kennedy Space Center (KSC) Education office for all the hard work done in organizing and facilitating the internship experience. Finally, I would like to give an extra thank you to Felix Soto Toro for being a great mentor.

References

[1] Lewis, M. and Wehner, W., "Advanced Inspection System (AIS) Hardware & Software Features – Intelligent Algorithms for Computer Vision & Image Processing Applications", KDDMS ACLO Share, Kennedy Space Center, Florida, 2014 (unpublished).

[2] Lewis, M., Wehner, W. and Wlodarczyk, S., "Advanced Inspection System (AIS) Software Developers Guide For Robot Operating System (ROS)", ACLO Intelligent Devices AIS SharePoint Site, Kennedy Space Center, Florida, 2014 (unpublished).