CyPhyHouse

Release 0.1

Oct 28, 2020

Project Links

This website is the technical documentation for the CyPhyHouse project. For a non-technical overview and research papers of the CyPhyHouse project, please visit our project website at https://cyphyhouse.github.io/.

Users are expected to familiarize themselves with Robot Operating System (ROS) and Gazebo simulation environment in order to use the software stack provide in CyPhyHouse project. We recommend beginners to at least walk through the tutorials in http://wiki.ros.org/ before trying out our software.

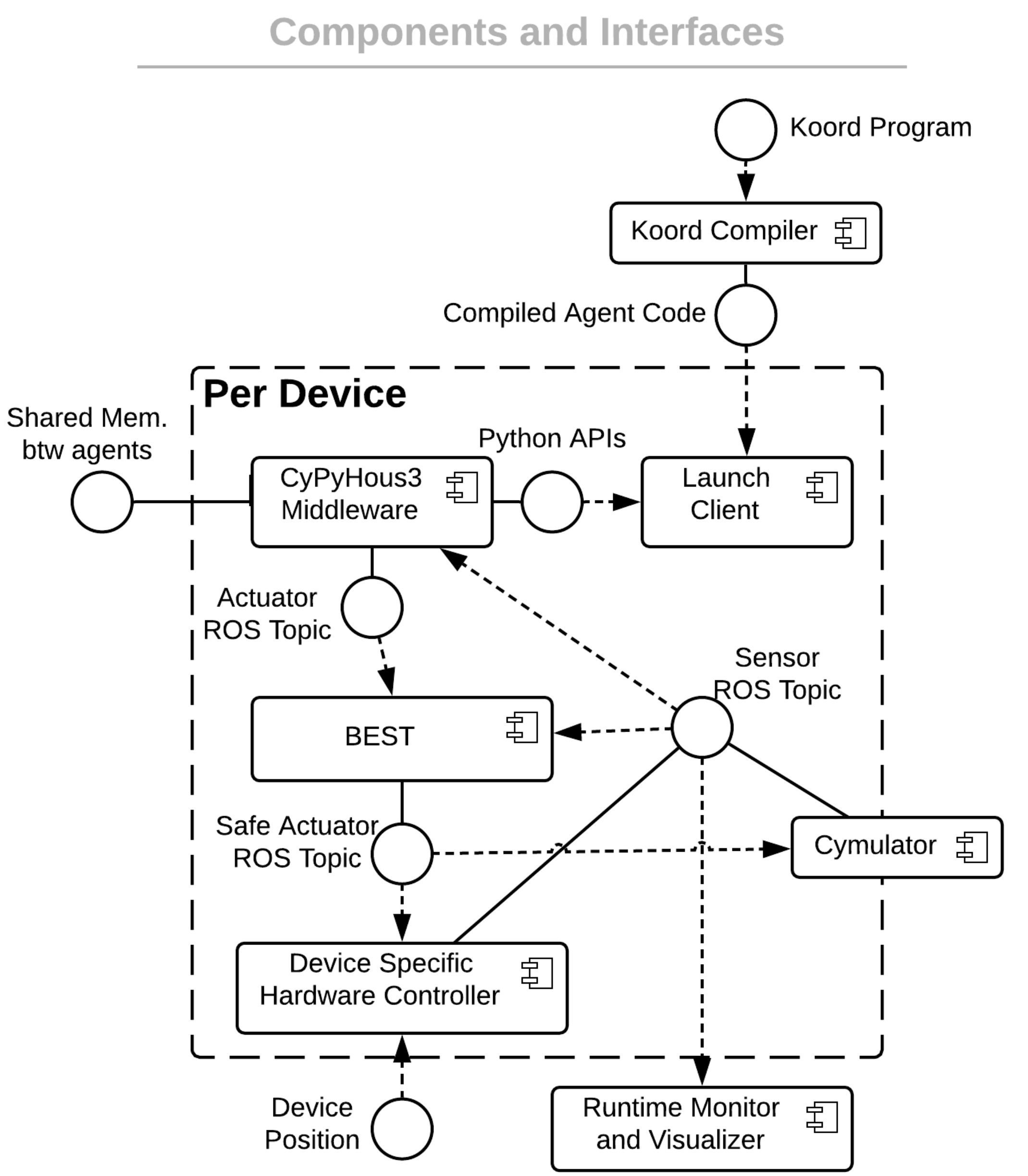
CHAPTER 1

Demo

Todo: Add demo videos and mention each component

1.1 Project Architecture

1.1.1 Software Components



1.1.2 Interfaces

Predefined ROS Topics

Shared between Simulation and Deployment

```
waypoint         geometry_msgs/PoseStamped
waypoint_tobest geometry_msgs/PoseStamped  # Not used yet
reached          std_msgs/String
```

Topic names are all lower cases by convention. Note that the topics are not global names. For deployment, we can specify environment variable `ROS_NAMESPACE` for each device so that `ros_launch` will append a prefix to the topics. That is the topic at runtime becomes `/{ROS_NAMESPACE}/waypoint`. `ROS_NAMESPACE` should be set with an unique ROS name such as IP with port.

In the simulator, `*.launch` is auto-generated. We explicitly generate a namespace for each simulated agent. E.g., the topic `waypoint` becomes `/{AGENTID}/waypoint` in simulation with the unique `AGENTID` as the namespace.

See Remapping Arguments for more detail.

Simulation only

Drone Specific

```
/drone{id}/ground_truth/state nav_msgs/Odometry
/drone{id}/cmd_vel            geometry_msgs/Twist
/drone{id}/goals              std_msgs/Float32MultiArray
```

Car Specific

```
/car{id}/racecar/left_rear_wheel_velocity_controller/command      std_msgs/Float64
/car{id}/racecar/right_rear_wheel_velocity_controller/command     std_msgs/Float64
/car{id}/racecar/left_front_wheel_velocity_controller/command     std_msgs/Float64
/car{id}/racecar/right_front_wheel_velocity_controller/command    std_msgs/Float64
/car{id}/racecar/left_steering_hinge_position_controller/command  std_msgs/Float64
/car{id}/racecar/right_steering_hinge_position_controller/command std_msgs/Float64

# goto.py
/car{id}/ground_truth/state nav_msgs/Odometry
/car{id}/goals              std_msgs/Float32MultiArray

# ackermann_car.py
/car{id}/ackermann_cmd      ackermann_msgs/AckermannDriveStamped
```

Deployment only

Vicon

```
/vrpn_client_node/{vicon_obj}/pose  geometry_msgs/PoseStamped
/vrpn_client_node/{vicon_obj}/twist geometry_msgs/TwistStamped
```

Drone Specific

```
/mavros/cmd/arming                mavros_msgs/CommandBool
/mavros/cmd/takeoff               mavros_msgs/CommandTOL
/mavros/cmd/land                  mavros_msgs/CommandTOL
/mavros/set_mode                  mavros_msgs/SetMode
/mavros/setpoint_position/local geometry_msgs/PoseStamped
/mavros/cmd/set_home              mavros_msgs/CommandHome
```

Car Specific

```
/ackermann_cmd ackermann_msgs/AckermannDriveStamped
```

See *Predefined ROS Topics*.

Koord Language Compiler

- Java + ANTLR

CyPyHous3 Middleware

- Python 3.5.2 + ROS + OMPL
- ROS for communication with hardware controllers
- OMPL for path planning

Hardware Controller

- C++ + ROS

Simulator and Visualizer

- Python 3.5.2 + Gazebo

Device Discovery and Launch

- Python 3.5.2

BEST Effort Safe Termination

- C++

Todo: Link to each repository? Version and packages for C++ and Java. Diagrams and interfaces?

1.1.3 Hardware Devices

F1/10

Drone

Todo: Fill in short descriptions for each device

1.2 Configuration Files

1.2.1 User specified Global Configurations

Example global configuration file

```
leader_pid: 1
mutex_handler: BaseMutexHandler
udp_bcast_ip: 127.255.255.255
udp_port: 61820
agents:
    - pid: 0
      on_device: drone0
      motion_automaton: MoatTestDrone
    - pid: 1
      on_device: drone1
      motion_automaton: MoatTestDrone
    - pid: 2
      on_device: drone2
      motion_automaton: MoatTestDrone
    - pid: 3
      on_device: hotdec_car
      motion_automaton: MoatTestCar

devices:
    hotdec_car:
      bot_type: CAR
      ip: 127.0.1.0
      ros_node_prefix: 'waypoint_node'
      queue_size: 1
      motion: &cym_moat_car  # Set anchor for reusing
        waypoint_topic:
            topic: 'waypoint'
            type: PoseStamped  # geometry_msgs/PoseStamped
        reached_topic:
            topic: 'reached'
            type: String  # std_msgs/String
        positioning_topic:
            topic:  '/vrpn_client_node/'  # TODO '"/vrpn_client_node/" + vicon_obj +
↪"/pose"'
            type: PoseStamped  # geometry_msgs/PoseStamped
        planner: SimplePlanner
        motion_automata: [MoatTestCar]

    f1car:
      bot_type: CAR
      ip: 127.0.1.1
      motion: *cym_moat_car  # Reuse car motion configs

    drone0:
      bot_type: QUAD
      ip: 127.0.2.0
      motion: &cym_moat_drone  # Set anchor for reusing
        ros_node_prefix: 'waypoint_node'
        queue_size: 1
        waypoint_topic:
            topic: 'waypoint'
```

(continues on next page)

(continued from previous page)

```
            type: PoseStamped  # geometry_msgs/PoseStamped
        reached_topic:
            topic: 'reached'
            type: String  # std_msgs/String
        positioning_topic:
            topic:  '/vrpn_client_node/'  # TODO '"/vrpn_client_node/" + vicon_obj +
↪"/pose"'
            type: PoseStamped  # geometry_msgs/PoseStamped
        planner: SimplePlanner
        motion_automata: [MoatTestDrone]

    drone1:
      bot_type: QUAD
      ip: 127.0.2.1
      motion: *cym_moat_drone  # Reuse car motion configs

    drone2:
      bot_type: QUAD
      ip: 127.0.2.2
      motion: *cym_moat_drone
```

1.2.2 Auto-generated Agent Local Configurations

Example local configuration file for one of the agents.

```
agent:
  motion_automaton: MoatTestDrone
  on_device: drone1
  pid: 1
device:
  bot_name: drone1
  bot_type: QUAD
  ip: 127.0.2.1
  motion_automata: [MoatTestDrone]
  planner: SimplePlanner
  port: 61820
  positioning_topic: {topic: /vrpn_client_node/, type: PoseStamped}
  queue_size: 1
  reached_topic: {topic: reached, type: String}
  ros_node_prefix: drone1/waypoint_node
  waypoint_topic: {topic: waypoint, type: PoseStamped}
leader_pid: 1
mutex_handler: BaseMutexHandler
num_agents: 4
udp_bcast_ip: 127.255.255.255
udp_port: 61820
```

Todo: Include usage of *gen_local_config* script to generate a local config from the global config

1.3 Koord Programming Framework

The Koord language is a new language for coordination in bots.

TODO briefly introduce and show example Koord code

1.3.1 Quick start using JAR package

Requirements

- Java Runtime Environment 11 (JRE 11) or above
- Download the JAR file `koord-*-jar-with-dependencies.jar` from one of our releases (or compile from source code)

Usage

Given a Koord program `app.krd`, run the following command to generate Python code `app.py`:

```
$ java -jar /path/to/koord-*-jar-with-dependencies.jar app.krd app.py
```

1.3.2 Compile JAR package from source code

Requirements

- Java Development Kit 12 (JDK 12)
- Maven
- Python 3.5 or above for testing

Compilation

The parser is written in Java and uses Antlr. This project uses Maven.

Run following command to build and test the JAR package file:

```
$ mvn package
```

The created JAR file should be under `target` folder following the name pattern `koord-*-jar-with-dependencies.jar`. With the JAR file, please follow the instructions in the previous section to run Koord compiler.

Syntax References

Koord is language that is focused on events and reacting to them. It uses significant whitespace similar to python.

A koord file consists of five main sections:

- Definitions
- Modules

- Variable Declarations
- Initiation
- Events

These sections must be declared in this order.

Definitions

The definitions blocks consists of function declarations.

Modules

The modules sections declares sensors and actuators that are to be used. Variables can either be an actuator or a sensor, must be declared in the respective block. Module names must begin with a capital letter. For instance, using the module Motion:

```
using Motion:
  actuators:
    pos target
  sensors:
    boolean done
```

Known Modules

Motion

```
using Motion:
  actuators:
    pos target
  sensors:
    boolean done
```

Log:

```
using Log:
  actuators:
    stream stdout
  sensors:
    stream  stdin
```

To use streams, the << syntax is needed.

```
stdout << "Hello World"
```

Variable Declaration

Variables must either be declared as local, allread or allwrite.

Variables need to have a type and must start with a lower case letter. Variables may also be given an initial value.

Local

Local means that a variable cannot be seen by other bots, it can only be seen by the bot with the variable.

Allread

allread means that other bots may read from the variable, but other bots may not write to the variable. The variable owner may still write to the bot. To declare an allread variable, it must be declared as an array. A read requires array access, with the index representing the id of the bot. An allread variable can only be written to by using the syntax `varname[pid] = ...`, and will not accept syntax that should be the same thing, such as `varname[pid * 1] = ...`

All Write

allwrite means any bot may write to the variable.

Example

```
allwrite:
  int a
  boolean b

allread:
  int[] c
  int[] d

local:
  int e
  float f
```

Events

Events consist of a label, a pre condition, and an effect. A precondition must be a boolean value. The precondution must be on the same line as the *pre:* label.

```
dosomething:
  pre:true
  eff:
    hello()
```

Types

- pos
- boolean
- int
- float
- stream

- arrays

Control Flow

Conditional

Koord supports `if` and `if else` statements. To use `elif`, do a nested `if else`.

Loops

Koord supports constant iteration `for` loops. Koord does not support while loops.

Example:

```
for i = 0, 5:
    doSomething()
```

Example Code

- Log
- Lineform
- Hvac
- Shapeform

Semantics

Distributed Shared Memory

When variables are dcelared `allread` and `allwrite`, they are in shared memory. All robots can read and write to `allwrite` variables and all robots can read from `allread` variables. `allread` variables need to be arrays. A robot can only write to one element of an allread variable.

Round based Execution

A program will find the first event that satisfies the precondition, execute it, then start from the top again.

Parser

The parser uses Antlr. Maven should look at the grammar file and generate the parser to *target/generate-sources*.

The parser then creates `KoordParser.<GrammarNode>Context` classes, which are used along with the `KoordBaseListener` class and the tree walker class to traverse the AST.

Generation

A Koord program is compiled into a python file.

There are two main components: initiation and events.

The initiation is handled by the function `initialize_vars` and contains setup for variables and the code for the `init` block.

The events are handled by the function `loop_body` and is meant to be called in a loop. It finds the first event that satisfies its precondition, executes the code and then returns.

Sensors and Actuators

Sensors and actuators get compiled to `self.read_from_sensor(sensor_name)` and `self.write_to_actuator(actuator_name, value)`, which inherit from the parent class.

Variables

Local variables get compiled to `self.locals[local_variable]`.

Shared variables require distributed memory, so they get compiled to calls to `self.write_to_shared(var_name, index, value)` and `self.read_from_shared(var_name, index)` which also inherit from the parent class allow it to do distributed memory stuff.

Others

Many other things, such as arithmetic operators and constants, are the same in both python and koord, and do not get transformed at all.

Example

Koord:

```
allwrite:
  int sum = 0
  int numadded = 0
local:
  boolean added = false
  int finalsum

adding:
  pre: !added
  eff :
      atomic:
         sum = sum + pid * 2
         numadded = numadded + 1
         added = true
finalsum:
  pre: numadded == numAgents
  eff :
      finalsum = sum
```

Generated Python:

```
from agentThread import AgentThread


class DefaultName(AgentThread):

    def __init__(self, config):
        super(DefaultName, self).__init__(config)
        self.start()

    def initialize_vars(self):
        self.locals = {}
        self.locals['added'] = False
        self.locals['finalsum'] = None
        self.create_aw_var('sum', int, 0)
        self.create_aw_var('numadded', int, 0)

    def loop_body(self):
        if not self.locals['added']:
            if not self.lock():
                return
            self.write_to_shared('sum', None, self.read_from_shared('sum', None) + self.pid() * 2)
            self.write_to_shared('numadded', None, self.read_from_shared('numadded', None) + 1)
            self.locals['added'] = True
            self.unlock()
            return
        if self.read_from_shared('numadded', None) == self.num_agents():
            self.locals['finalsum'] = self.read_from_shared('sum', None)
            return
```

Program Analysis

Control Flow Graph

Control flow is handled by the class `BasicBlock`. It has two outgoing arrows if it ends in a conditional statement, and one outgoing arrow if it is a "merge" block.

Timing Analysis

Timing analysis is done by the algorithm `worstPath(block) = cost(block) + max(worstPath(block.left), worstPath(block.right))` with base case being `worstPath(block) = cost(block)` if it is a leaf node.

1.4 Cymulator: ROS-Gazebo based Simulator

1.4.1 Installation

The installation steps below are also assembled in this shell script that should work for Ubuntu 16.04. These commands requires *sudo* permission. Please run them with caution.

1. Install ROS Kinetic and create a workspace for catkin. We assume it is under *catkin_ws*.
   - ROS Kinetic Ubuntu
   - Creating a workspace for catkin
2. Install Gazebo 9 for ROS Kinetic

```
sudo sh -c 'echo "deb http://packages.osrfoundation.org/gazebo/ubuntu-stable `lsb_
↪release -cs` main" > /etc/apt/sources.list.d/gazebo-stable.list'
wget http://packages.osrfoundation.org/gazebo.key -O - | sudo apt-key add -
sudo apt-get update
sudo apt install -y \
     ros-kinetic-gazebo9-ros ros-kinetic-gazebo9-ros-control \
     ros-kinetic-gazebo9-plugins ros-kinetic-gazebo9-ros-pkgs
```

3. Install required ROS packages available on APT

```
sudo apt install -y \
     ros-kinetic-ackermann-msgs ros-kinetic-geographic-msgs ros-kinetic-serial \
     ros-kinetic-ros-control ros-kinetic-ros-controllers \
     ros-kinetic-hector-localization ros-kinetic-hector-models \
     ros-kinetic-geometry2 ros-kinetic-robot
```

4. Install other system packages available on APT

```
sudo apt install -y git
sudo apt install -y cppad coinor-libipopt-dev  # For MPC controller
sudo apt install -y python3 python3-pip
```

5. Install required Python packages available on PyPI

```
pip3 install --user pip --upgrade
pip3 install --user \
     catkin_pkg rospkg \
     empy numpy scipy \
     defusedxml netifaces \
     pathlib pyyaml
```

6. Inside the *catkin_ws/src* directory of your catkin workspace clone the following repos:

```
git clone https://github.com/tu-darmstadt-ros-pkg/hector_quadrotor.git --branch␣
↪kinetic-devel
git clone https://github.com/tu-darmstadt-ros-pkg/hector_gazebo.git --branch␣
↪kinetic-devel
git clone https://github.com/cyphyhouse/racecar.git --branch RacecarJTransitory
git clone https://github.com/cyphyhouse/racecar_gazebo.git --branch master
git clone https://github.com/cyphyhouse/Decawave.git --branch for-cymulator
git clone https://github.com/cyphyhouse/Cymulator.git --branch master
```

Compile using catkin_make

7. Run these commands under your *catkin_ws* directory to compile relevant ROS packages in the cloned repositories.

```
source /opt/ros/kinetic/setup.bash
catkin_make --only-pkg-with-deps cym_gazebo --cmake-args -DPYTHON_VERSION=3.5  #␣
↪Build only cym_gazebo with Python>=3.5
```

Compile using colcon

7. Run these commands under your *catkin_ws* directory to compile only relevant ROS packages in *catkin_ws/src*.

```
source /opt/ros/kinetic/setup.bash
colcon build --base-paths src/* --packages-up-to cym_gazebo --cmake-args -DPYTHON_
↪VERSION=3.5
```

1.4.2 Usage

```
rosrun cym_gazebo cymulate.py scenes/empty-1_car.yml
```

Todo: Add sample YAML file for scenes and explain shell commands that start the Gazebo simulation