A Brief Introduction to ROS

About this Tutorial

- What software framework does a robot need?
- How does ROS fulfill these requirements?
- A quick demo

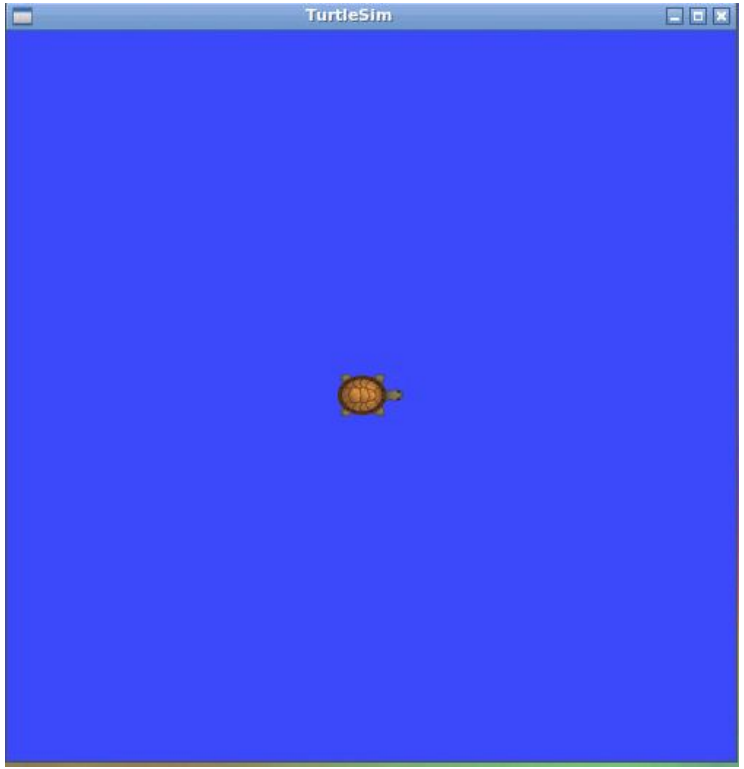
A Minimalistic Robot: (Simulated) Turtlebot

Sensors

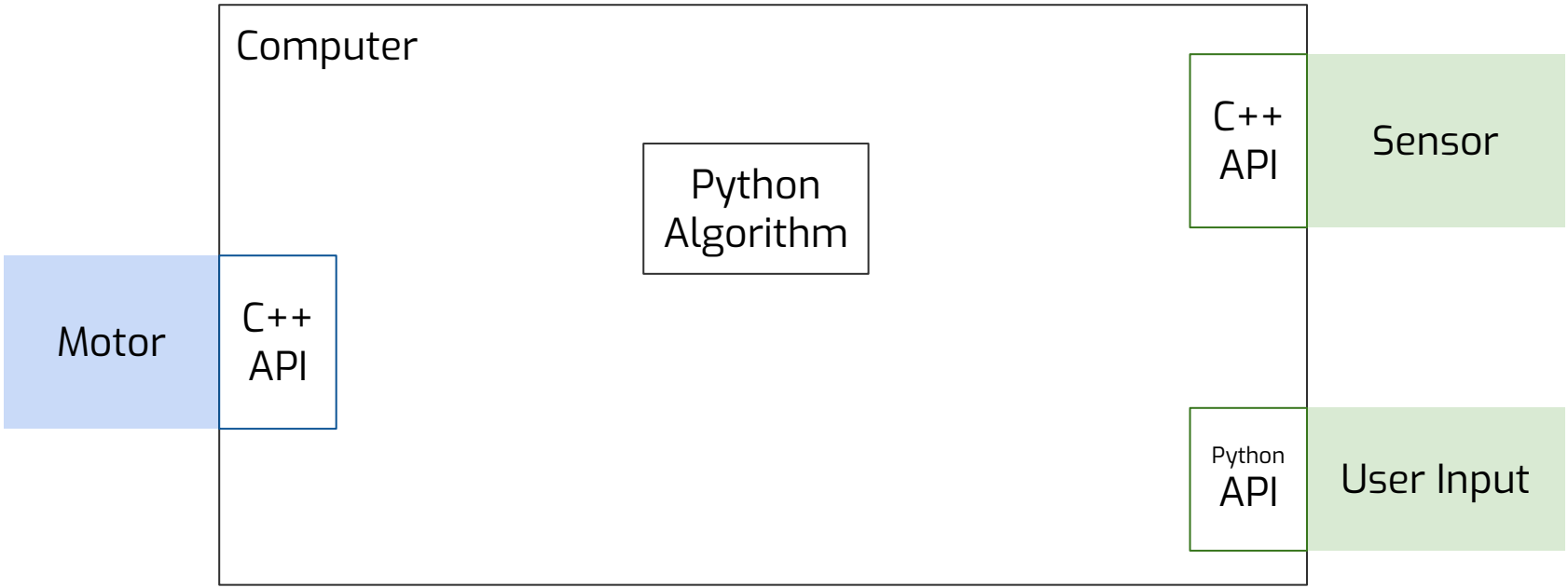
- Position
- Orientation
- User interface
  e.g., keyboard, touchscreen, joystick

Actuators

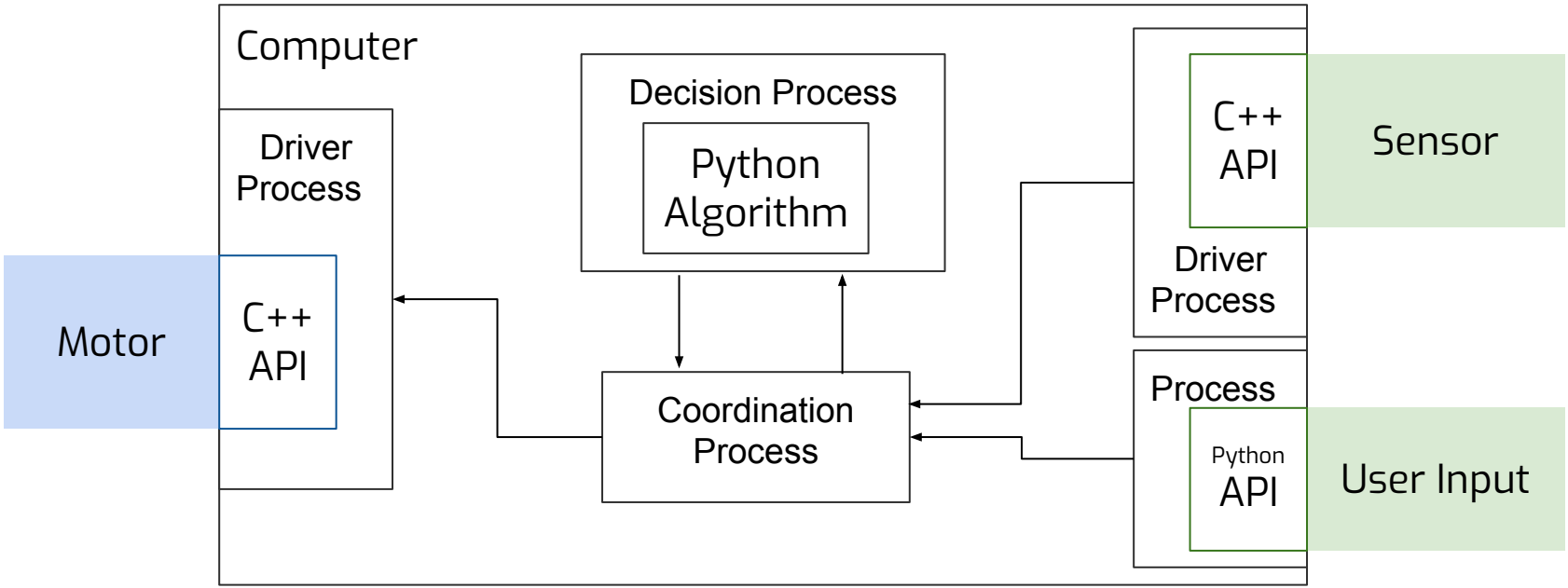
- Velocity



A Minimalistic Robot: (Simulated) Turtlebot



One Architecture



Functionalities for Robotics

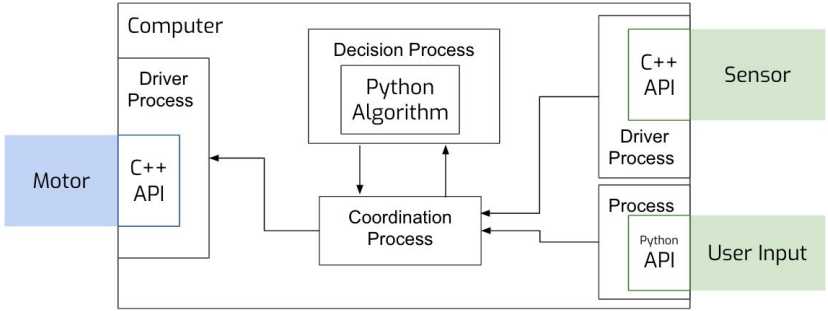
- Interprocess communication
  Passing data among processes (potentially on different computers)

- Data serialization and deserialization
  Translate data between Python and C++ processes

```
data = MyData()
data.field1 = 1
data.field2 = 2
```

```
MyData data;
data.field1 = 1;
data.field2 = 2;
```

Functionalities for Robotics



- Interprocess communication
  Passing data among processes

- Data serialization and deserialization
  Translate data between Python and C++ processes

- Logging
- Starting processes (from a configuration)
- Packages/Libraries management
- ...

One-Stop Solution: Robot Operating System (ROS)

- ROS topics, services, and action servers (only C++ and python)
  - Interprocess communication
  - Data serialization and deserialization
- roslog, rosbag: Logging
- rosrun, roslaunch: Starting processes (from a configuration)
- rosdep, rospack: Packages/Libraries management
- rosparam: Data accessible and editable from all processes
- catkin: Build system for both C++ and python
- …

One-Stop Solution: Robot Operating System (ROS)

```
ir@9ddf6466b1c0:~$ ros
rosawesome                      rosdistro_migrate_to_rep_143  rospack
rosbag                          rosdistro_reformat            rospair
rosboost-cfg                    rosed                         rosparam
roscat                          rosgraph                      rospd
roscd                           rosinstall                    rospython
rosclean                        rosinstall_generator          rosrun
rosco                           roslaunch                     rosservice
rosconsole                      roslaunch-complete            rossrv
roscore                         roslaunch-deps                rosstack
roscp                           roslaunch-logs                rostest
roscreate-pkg                   roslocate                     rostopic
rosd                            rosls                         rosunit
rosdep                          rosmake                       rosversion
rosdep-source                   rosmaster                     rosws
rosdistro_build_cache           rosmsg                        roswtf
rosdistro_freeze_source         rosmsg-proto
rosdistro_migrate_to_rep_141    rosnode
ir@9ddf6466b1c0:~$
```

Let’s Go Through A Few ROS Concepts!

- ROS packages
- ROS master process (`roscore`)
- ROS nodes, rosrun, roslaunch
- ROS topics and services

ROS Package

- A collection of libraries, executables, configurations, and data.
- Install, e.g., `sudo apt-get install ros-kinetic-turtlesim`
  (ROS package naming: `ros-{ros version}-{package name}`)
- Develop your own or build from source (ROS tutorial)
- ROS tools:
  - Build: find libraries
  - Launch: start process from executables
  - Data: ROS C++/Python APIs can locate the data

ROS Package

Try `rospack list`

```
ir@15e48840d0da:~$ rospack list
actionlib /opt/ros/kinetic/share/actionlib
actionlib_msgs /opt/ros/kinetic/share/actionlib_msgs
actionlib_tutorials /opt/ros/kinetic/share/actionlib_tutorials
amcl /opt/ros/kinetic/share/amcl
angles /opt/ros/kinetic/share/angles
astra_camera /opt/ros/kinetic/share/astra_camera
astra_launch /opt/ros/kinetic/share/astra_launch
base_local_planner /opt/ros/kinetic/share/base_local_planner
bfl /opt/ros/kinetic/share/bfl
bond /opt/ros/kinetic/share/bond
bondcpp /opt/ros/kinetic/share/bondcpp
bondpy /opt/ros/kinetic/share/bondpy
camera_calibration /opt/ros/kinetic/share/camera_calibration
camera_calibration_parsers /opt/ros/kinetic/share/camera_calibration_parsers
camera_info_manager /opt/ros/kinetic/share/camera_info_manager
capabilities /opt/ros/kinetic/share/capabilities
carrot_planner /opt/ros/kinetic/share/carrot_planner
catkin /opt/ros/kinetic/share/catkin
class_loader /opt/ros/kinetic/share/class_loader
clear_costmap_recovery /opt/ros/kinetic/share/clear_costmap_recovery
cmake_modules /opt/ros/kinetic/share/cmake_modules
collada_parser /opt/ros/kinetic/share/collada_parser
```

ROS Package

Try `rospack find turtlesim`
(Install via `sudo apt-get install ros-kinetic-turtlesim`)

```
ir@15e48840d0da:~$ rospack find turtlesim
/opt/ros/kinetic/share/turtlesim
```

ROS Master Process

- A background process
- Coordinates ROS communications
- Can be manually started by the executable `roscore`

ROS Master Process

Run `roscore`

```
ir@15e48840d0da:~$ roscore
... logging to /home/ir/.ros/log/404616d2-93e0-11ed-afe5-0242ac11
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://15e48840d0da:34007/
ros_comm version 1.12.17


SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.17
```

ROS nodes, rosrun, roslaunch

- Start an executable
  `rosrun {pacakge_name} {executable_name}`
  `rosrun kortex_driver kortex_driver`

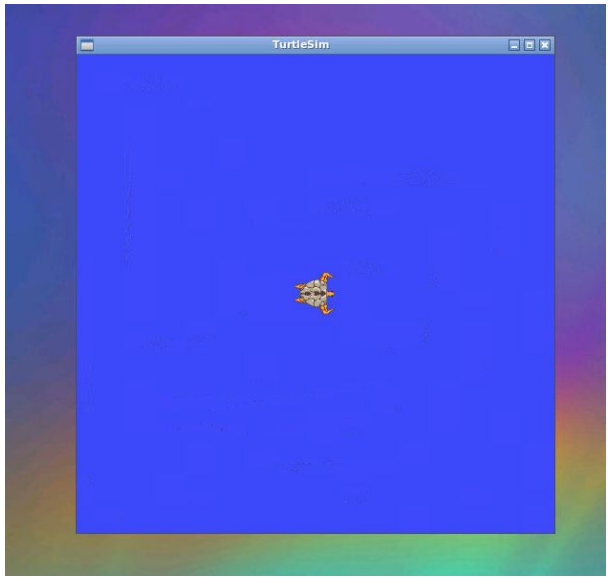
- Start multiple processes using a configuration
  `roslaunch {package_name} {config_file_name}`
  `roslaunch my_application scenario_1.launch`

- ROS figures out where to find them
- A ROS node is an executable registered with ROS master process/node
  `rospy.init_node(“my_node_default_name”)`

ROS nodes, rosrun, roslaunch

Run `DISPLAY=:1.0 rosrun turtlesim turtlesim_node`

Run `rosnode list`




```
ir@15e48840d0da:~$ rosnode list
/rosout
/turtlesim
```

ROS Interprocess Communication

Two patterns:

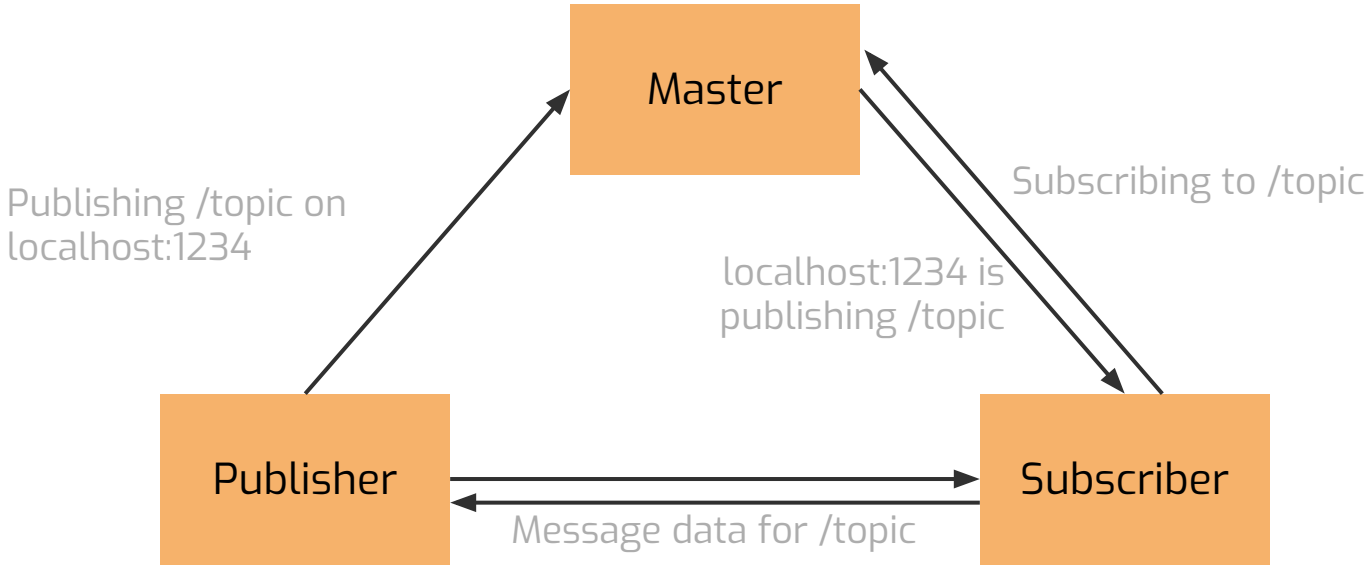
- ROS topics: Publisher-Subscriber

- ROS services: Request-Response

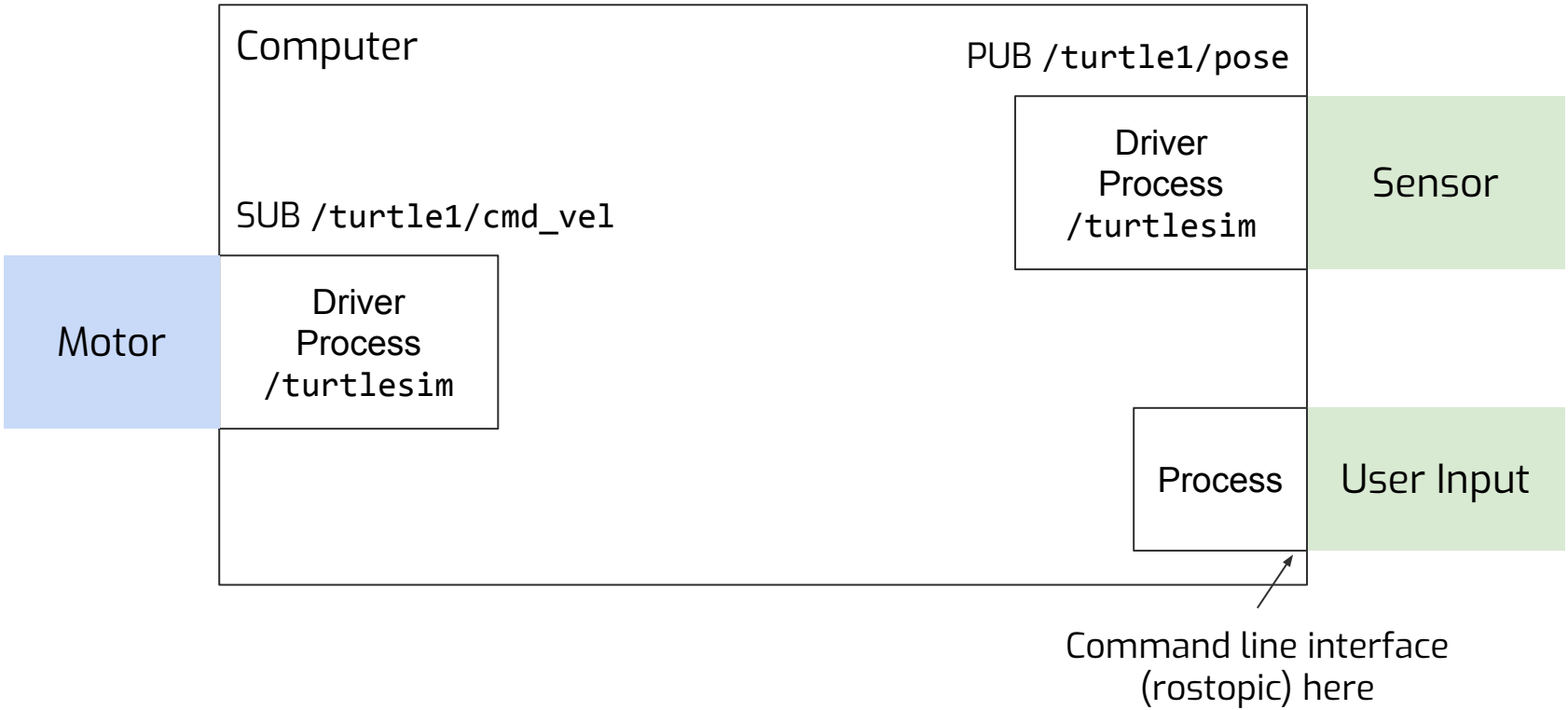
ROS Topics

- Topic: identifiable by a unique string
- Publisher (PUB)
  - Sends data to that topic
  - Does not care after sending the data
- Subscriber (SUB)
  - Listens that topic
  - Processes data with a callback
- Message: Data format definition

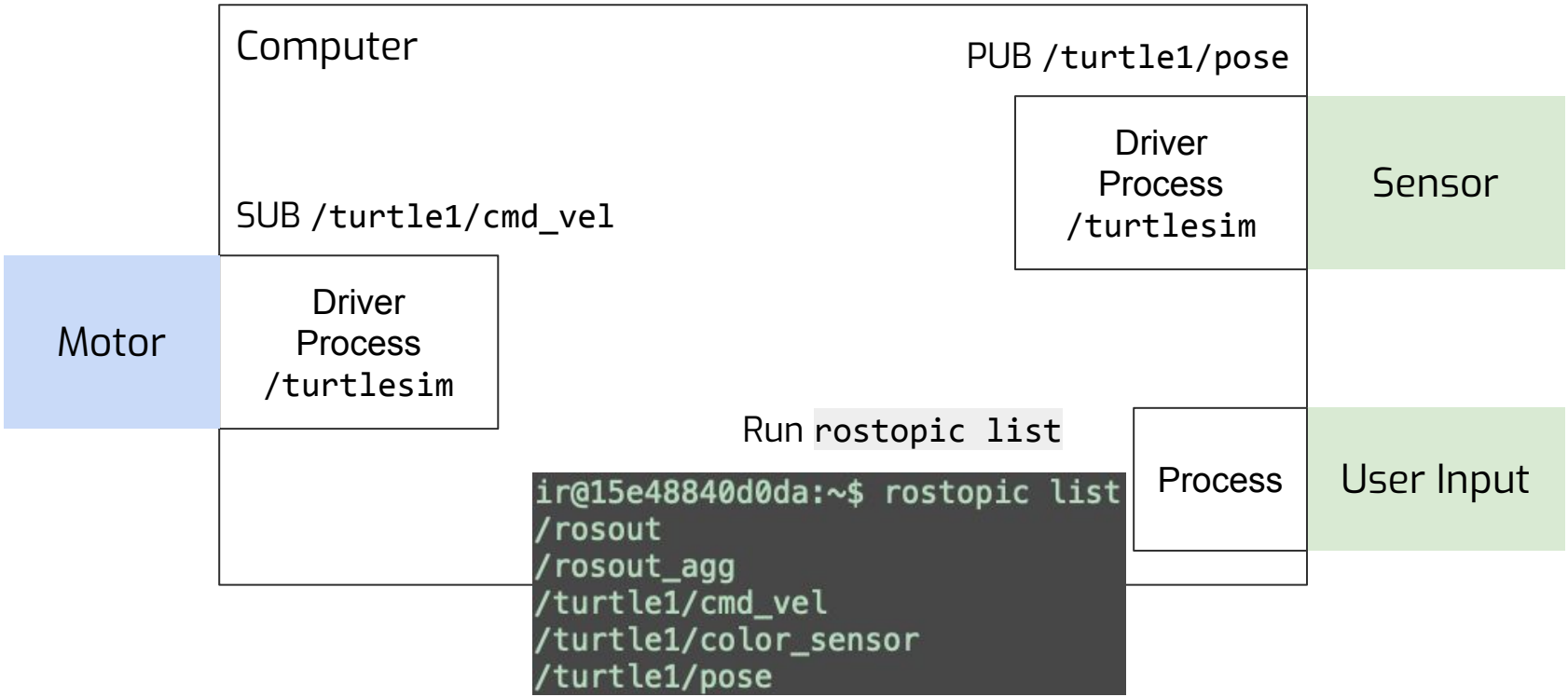
ROS Topics: Coordinated by ROS Master



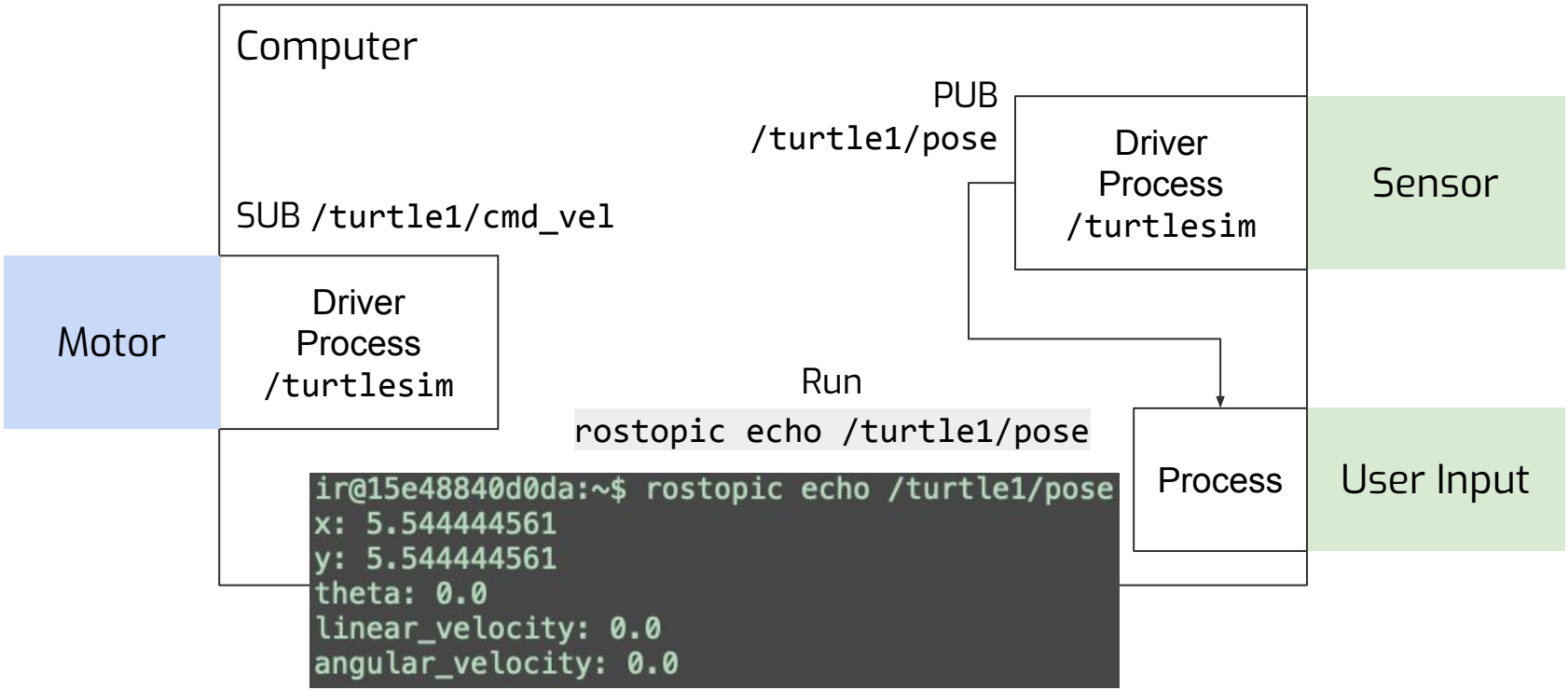
ROS Topics: Turtle Robot



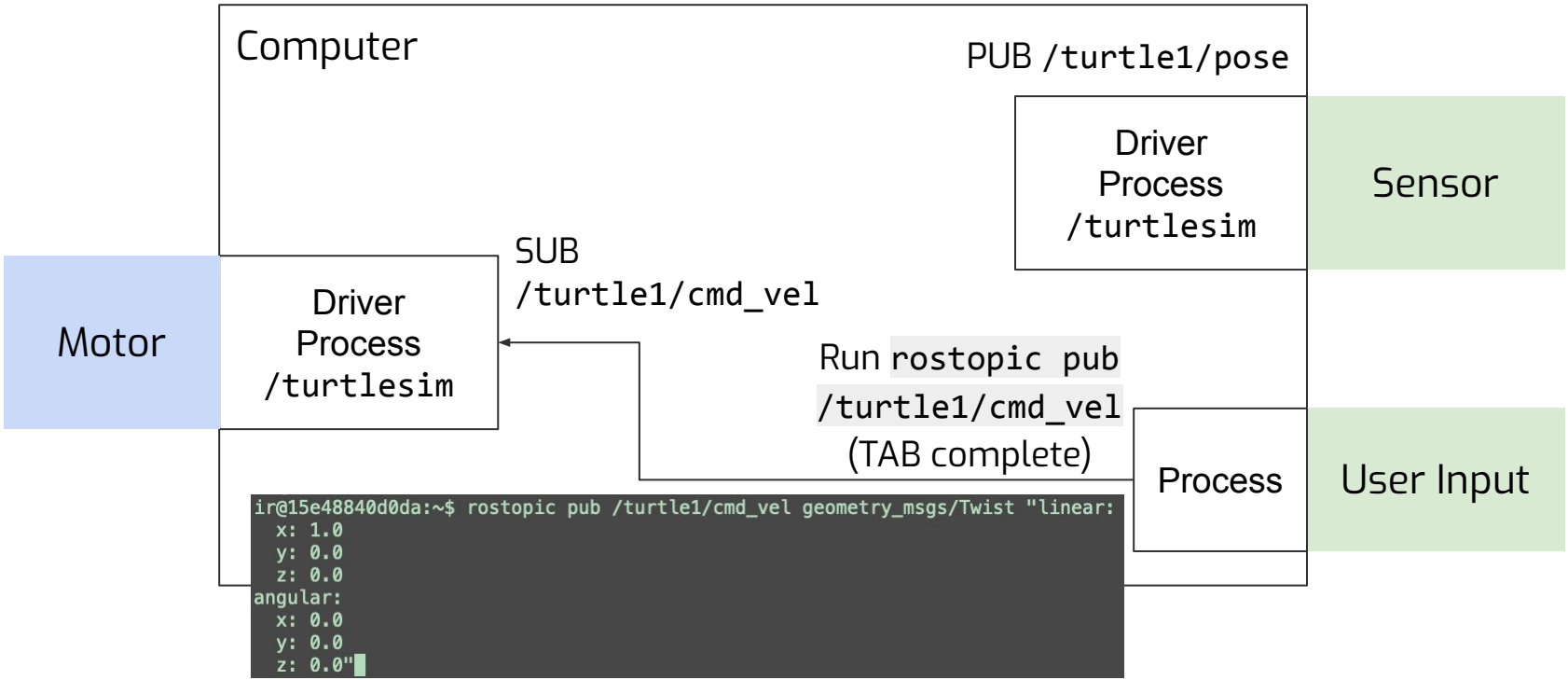
ROS Topics: List Topics



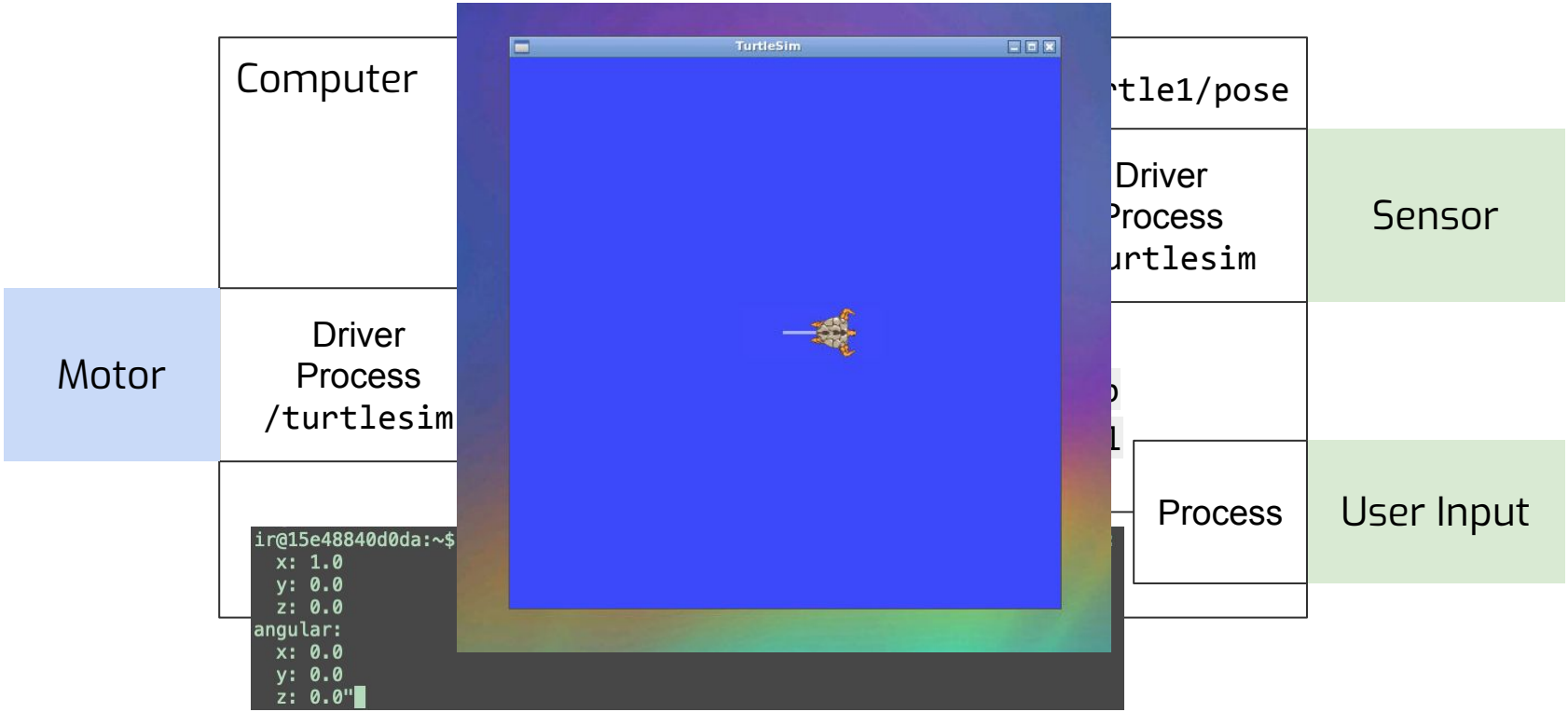
ROS Topics: Echo/Subscribe to A Topic



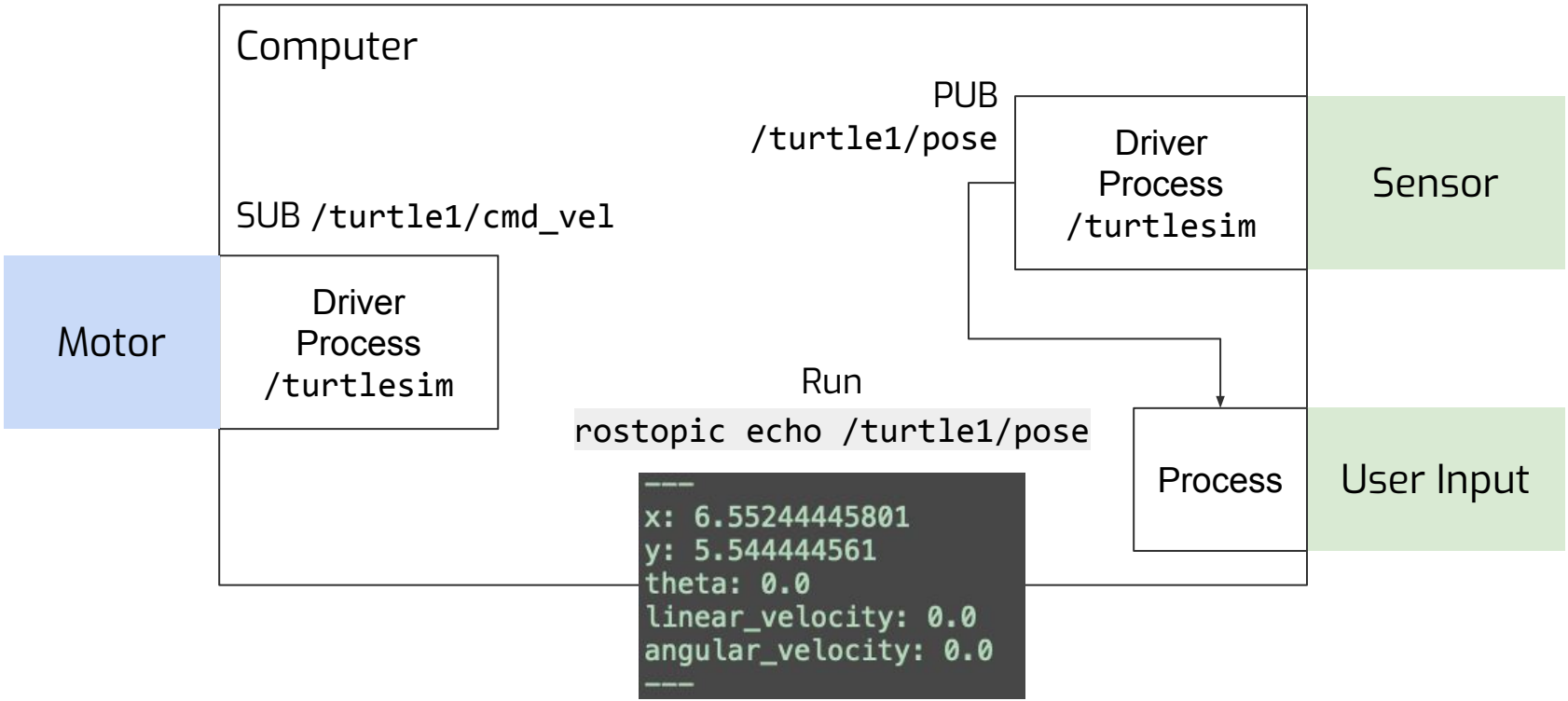
ROS Topics: Publish to A Topic



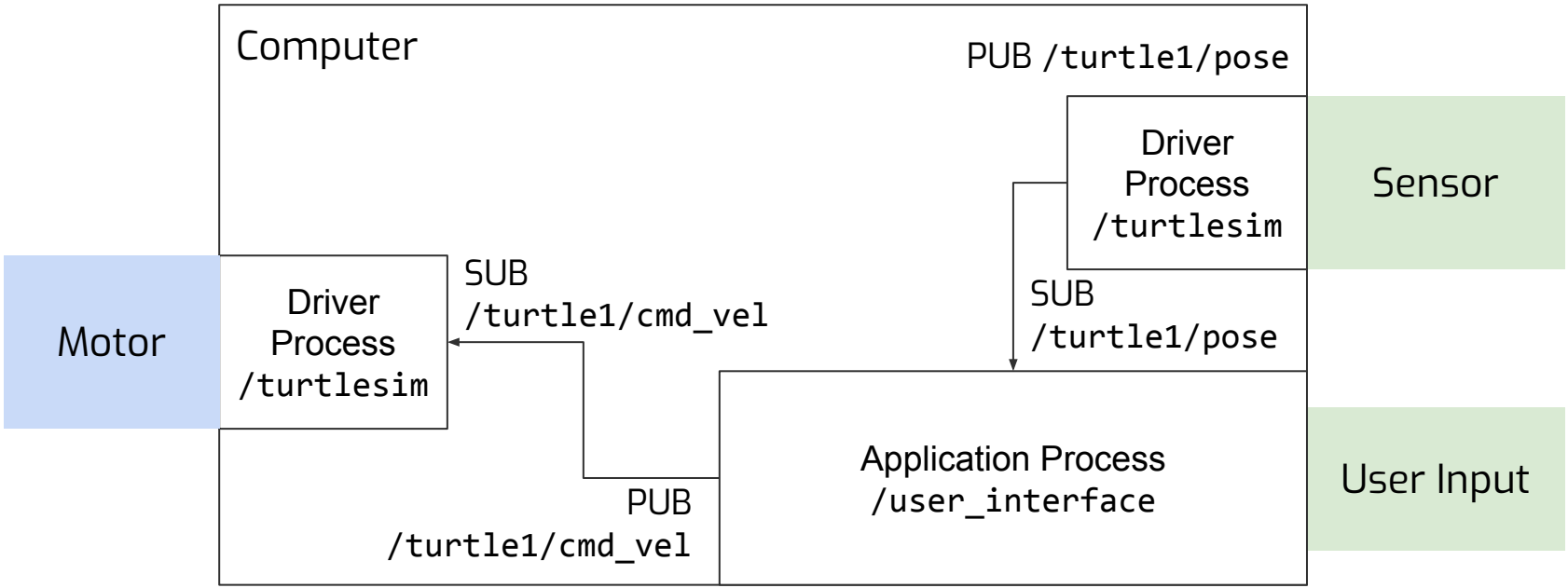
ROS Topics: It Moves!



ROS Topics: It Moves!



ROS Topics: Develop An “Application”



ROS Topics: Develop An “Application”

The “Algorithm”

```python
class Controller:
    """A simple controller/policy."""

    def __init__(self, x_goal, y_goal):
        self._goal = np.array([x_goal, y_goal])

    def __call__(self, x, y, theta):
        e = self._goal - np.array([x, y])
        e_dist = np.linalg.norm(e)
        e_theta = np.arctan2(e[1], e[0]) - theta
        e_theta = np.arctan2(np.sin(e_theta), np.cos(e_theta))
        # Treat distance and angle as two independent first-order systems
        v = min(e_dist, 1)  # clamp/saturation
        w = 2 * e_theta
        return v, w

    @property
    def goal(self):
        return self._goal
```

Algo input format

Algo output format

ROS Topics: Develop An “Application”

Wrapped ROS subscriber: format sensor data.

```python
class PositionSubscriber:
    """Subscriber to the robot's sensor."""

    def __init__(self):
        # Instantiate the ROS subscriber
        self._listener = rospy.Subscriber("/turtle1/pose", Pose, self._callback)

    def _callback(self, msg):
        """Callback function to store relevant data."""
        self.data = (msg.x, msg.y, msg.theta)
```

“Sensor” format

Algo input format

ROS Topics: Develop An “Application”

Wrapped ROS publisher: format command data.

```python
class CmdPublisher:
    """Publisher to the robot's actuator."""

    def __init__(self):
        # Instantiate the ROS publisher
        self._pub = rospy.Publisher("/turtle1/cmd_vel", Twist, queue_size=1)

    def __call__(self, v, w):
        """Format and then publish the ROS message."""
        # Put the data into the proper ROS message format
        msg = Twist()  # A ROS message
        msg.linear.x = v
        msg.angular.z = w
        # Publish the data using a ROS publisher
        self._pub.publish(msg)
```

Algo output format

“Actuator” format

ROS Topics: Develop An “Application”

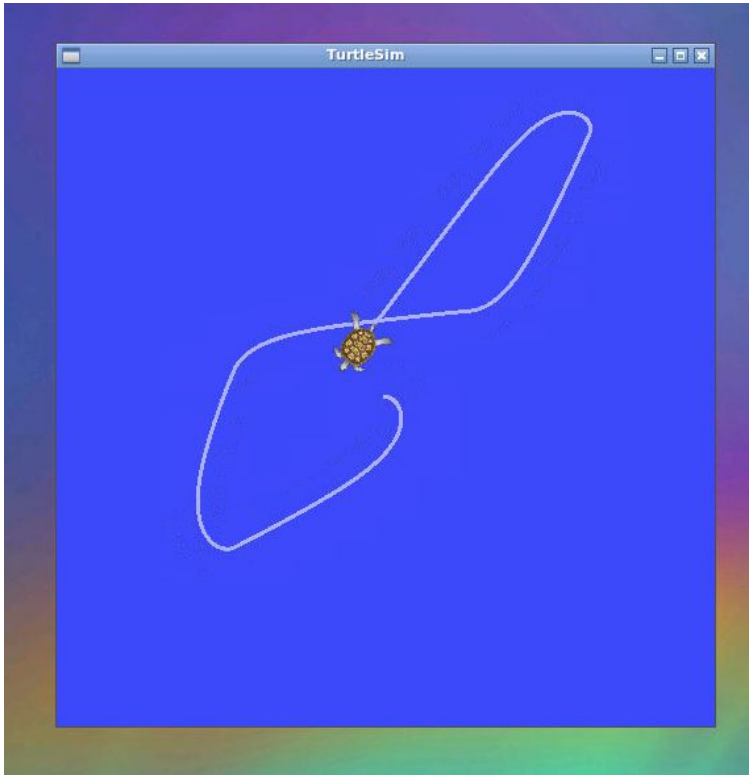
```
if __name__ == "__main__":
    rospy.init_node("user_interface")
    listener = PositionSubscriber()
    publisher = CmdPublisher()
    while not rospy.is_shutdown():
        user_input = raw_input(
            " q\t quit\n r\t reset\n x,y\t set goal to (x, y)\nYour command: "
        )
        if user_input == "q":
            sys.exit(0)
        if user_input == "r":
            client = rospy.ServiceProxy("/reset", std_srvs.srv.Empty)
            client.call()
        else:
            strings = user_input.split(",")
            new_goal = [float(s) for s in strings]
            controller = Controller(*new_goal)
            t0 = time.time()
            while not rospy.is_shutdown():
                v, w = controller(*listener.data)
                publisher(v, w)
                rospy.sleep(0.01)
                current_pos = np.array(listener.data[:2])
                if np.linalg.norm(current_pos - controller.goal) < 0.2:
                    rospy.loginfo("Goal reached")
                    break
                if time.time() - t0 > 5:
                    rospy.logwarn("Timeout reaching the goal!")
                    break
```

Sensor & actuator interfaces

“Algorithm”

Sensing-Reasoning -Acting loop

ROS Topics: Develop An “Application”




```
ir@15e48840d0da:~/misc$ python turtlesim_goto.py
 q       quit
 r       reset
 x,y     set goal to (x, y)
Your command: 3,3
[INFO] [1673687418.215339]: Goal reached
 q       quit
 r       reset
 x,y     set goal to (x, y)
Your command: 3,6
[INFO] [1673687430.615488]: Goal reached
 q       quit
 r       reset
 x,y     set goal to (x, y)
Your command: 7,7
[INFO] [1673687440.417882]: Goal reached
 q       quit
 r       reset
 x,y     set goal to (x, y)
Your command: 9,10
[INFO] [1673687451.979568]: Goal reached
 q       quit
 r       reset
 x,y     set goal to (x, y)
Your command: 1,1
[WARN] [1673687459.112277]: Timeout reaching the goal!
 q       quit
 r       reset
 x,y     set goal to (x, y)
Your command:
```

ROS Services

- Service: identifiable by a unique string

- Server:
  - Receives request and processes it with a callback.
  - Returns response.

- Client:
  - Sends request; receives response.
  - Waits for response (block).

- Example: reset the simulator `rosservice call /reset "{}"`
  Don’t expect this when working on real robots ;-)

ROS Interprocess Communication

Two patterns:

- ROS topics: Publisher-Subscriber
  - Sender does not care

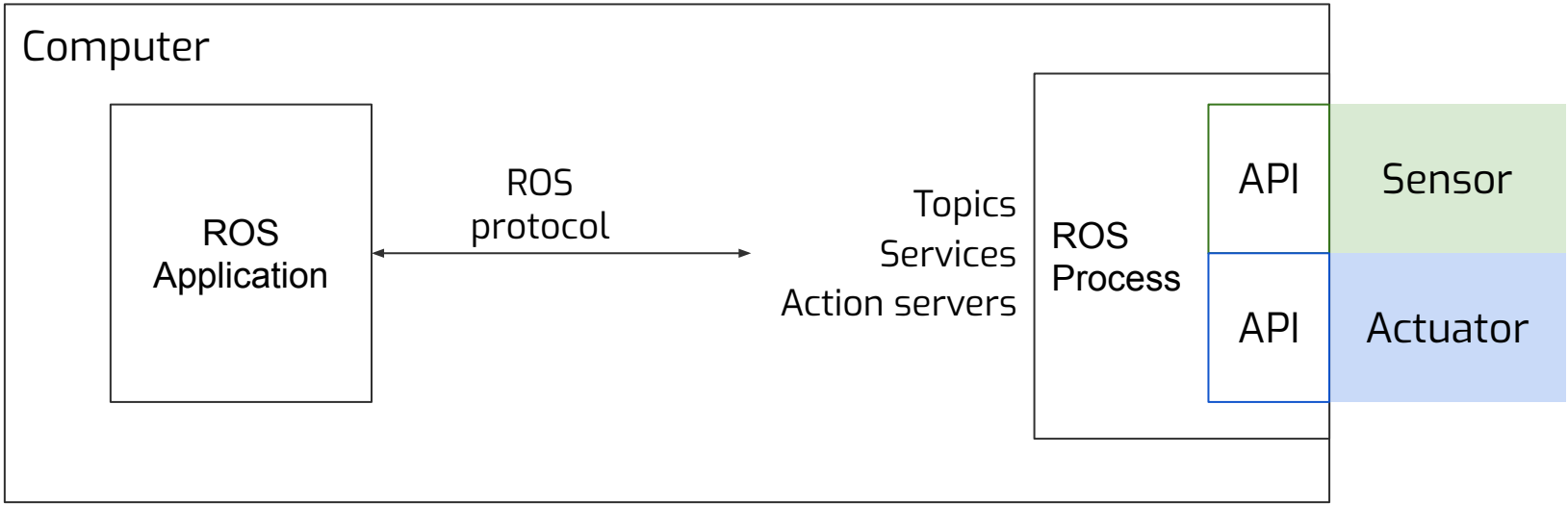
- ROS services: Request-Response
  - Sender cares

- ROS action server: Request-Response-Feedback
  - Similar to services
  - When a request takes a long time

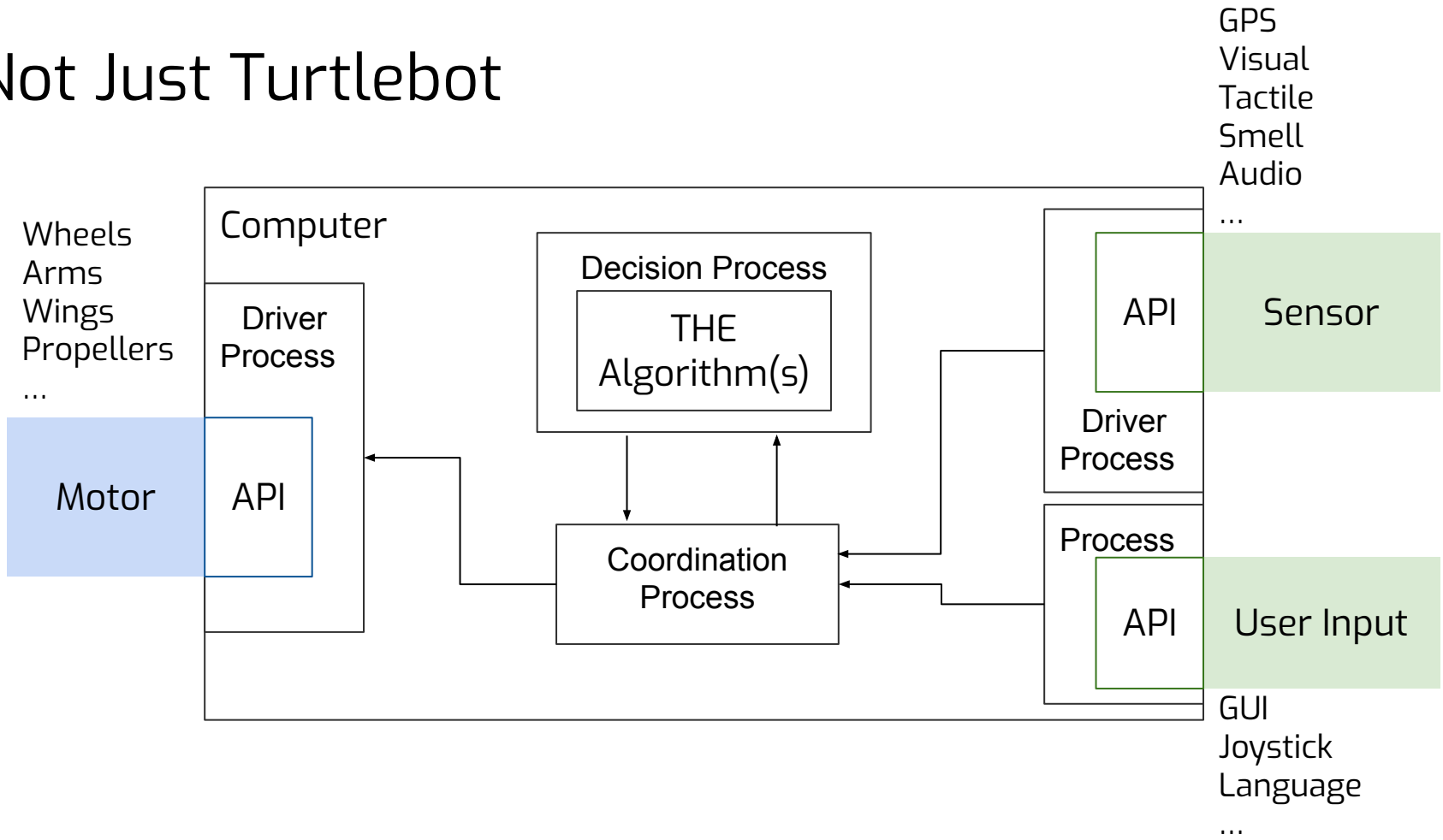
“ROS Drivers”

Some hardware vendors provide “ROS drivers”

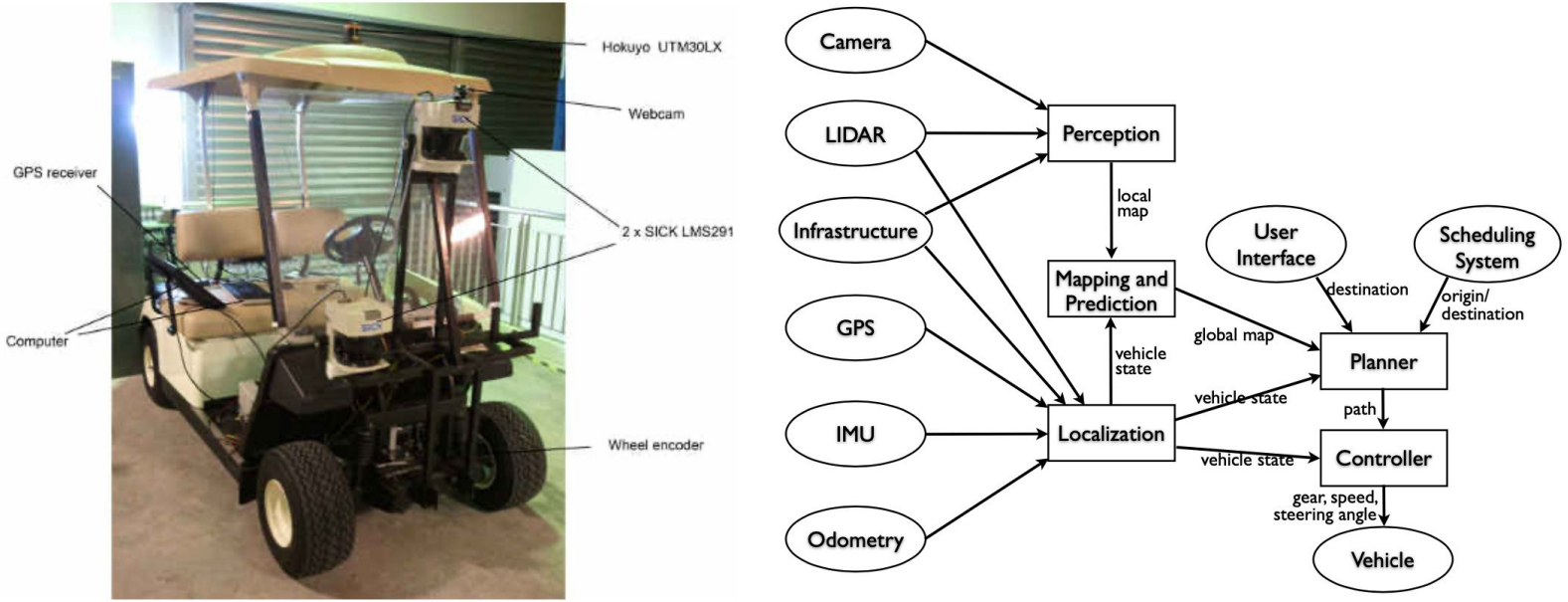
- Wrap their APIs with ROS interfaces



Not Just Turtlebot



ROS in the autonomous car



(a) System Testbed (b) System Architechture

Fig. 1. Autonomous Vehicle Testbed

Chong, Z. J. "Autonomous personal vehicle in crowded campus environments.

Advantages and Disadvantages of ROS

Advantages

- Provides lots of infrastructure, packages, and capabilities
- Easy to try other people's work and share your own
- Large community
- Free

Great for open-source and researchers.

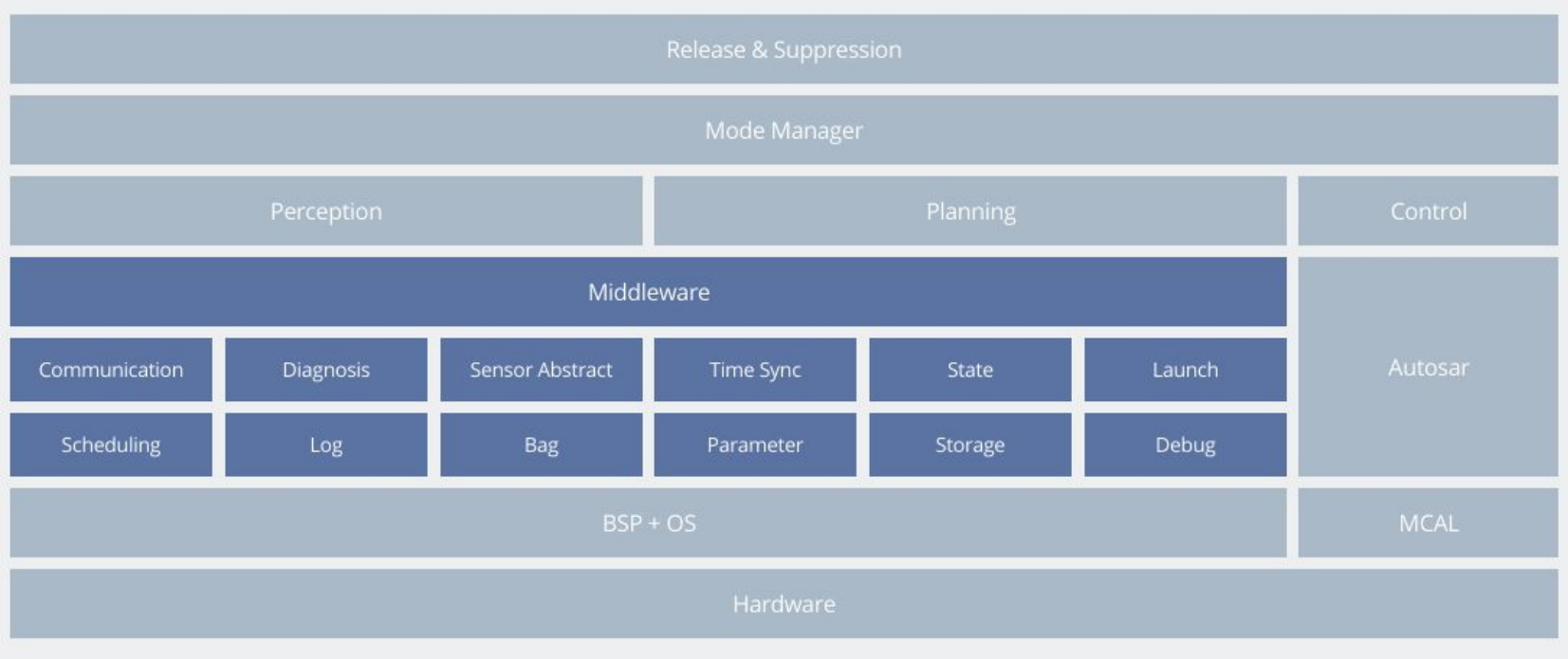
Disadvantages

- Approaching maturity, but still changing
- Security and scalability are not first-class concerns
- OSes other than Ubuntu Linux are not well supported
- Not real-time

Not great for mission-critical tasks and industry use.

Build You Own Wheels?

(DJI Auto's In-House Software)



Take-Home Message

- ROS: one-stop solution for robotics
- Not the only/optimal solution
- Many hardwares/libraries come with (suboptimal) ROS interfaces/drivers
  - Ideal for research/prototyping

More Tutorials

- http://wiki.ros.org/ROS/Tutorials

- turtlesim - ROS Wiki Video tutorials

Acknowledgement

- Part of the slides follow Justin Huang's ROS tutorial

- Part of the examples are taken from ROS kinetic official tutorial