# ROS2 Tutorial

## Robot Operating System

Made by Matthew Vong
Adapted from Justin Huang
https://youtu.be/bJB9tv4ThV4

# Basic Terminology

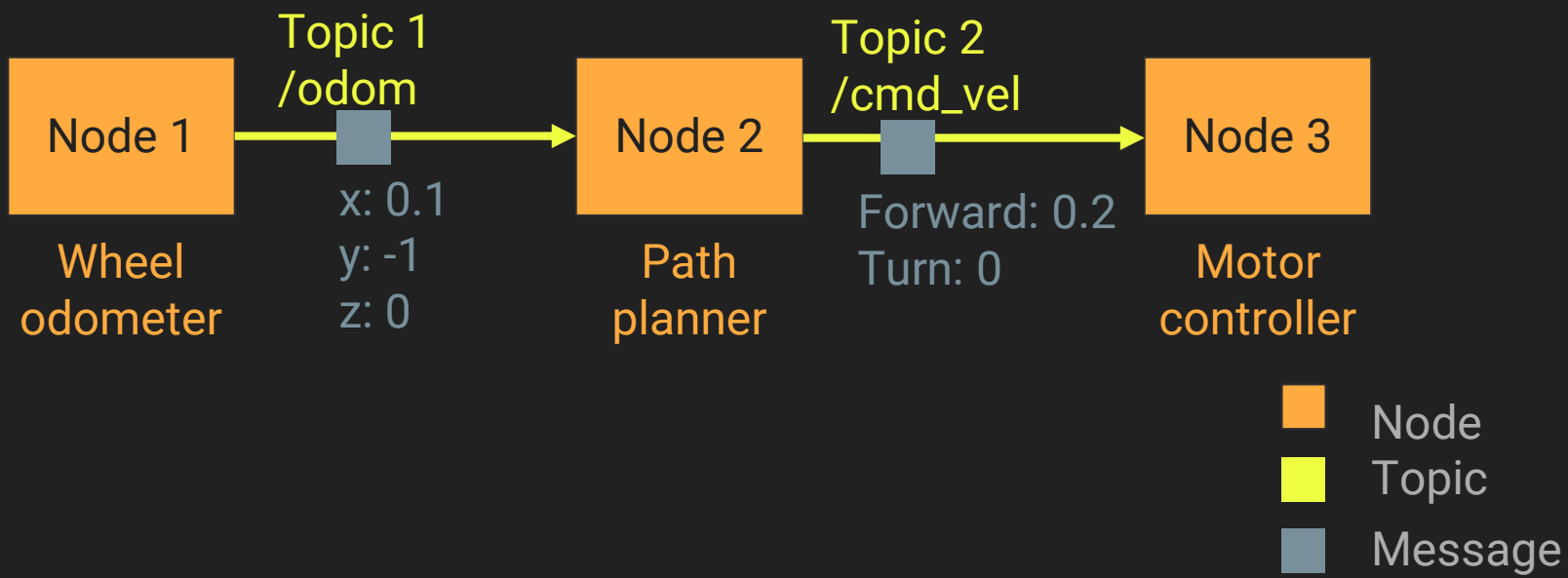
- Package: a collection of nodes to perform some task
- Node: a process in charge of only one specific task
- Topic: channels used to exchange data between nodes

Software in ROS is organised into packages
Packages may consist of many nodes
Nodes communicate with each other using topics
Topics are used to exchange structured data (messages)

# Why ROS?

- Aim: make code more modular by using nodes
- Benefits:
  - Code is easier to debug since each node only does one task
  - Open source and lots of existing packages available
  - Great simulations tools e.g. rviz2 and gazebo
  - Language agnostic: nodes can be written in C++ or python and can even communicate across different machines!

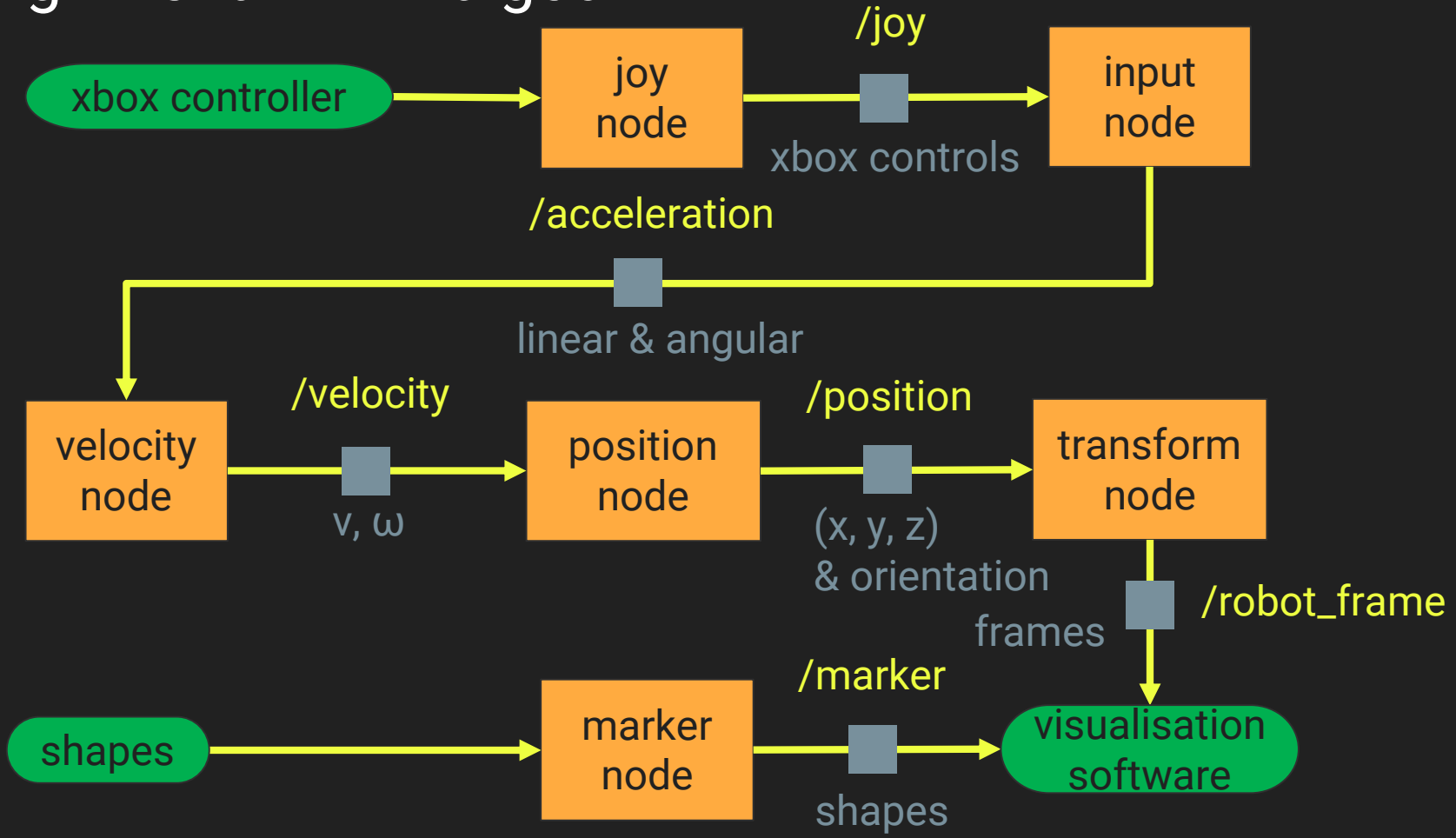
# ROS computation graph



# ROS uses a publisher-subscriber model

- Any node can publish a message to multiple topics
- Any node can subscribe to multiple topics
- Multiple nodes can publish to the same topic
- Multiple nodes can subscribe to the same topic

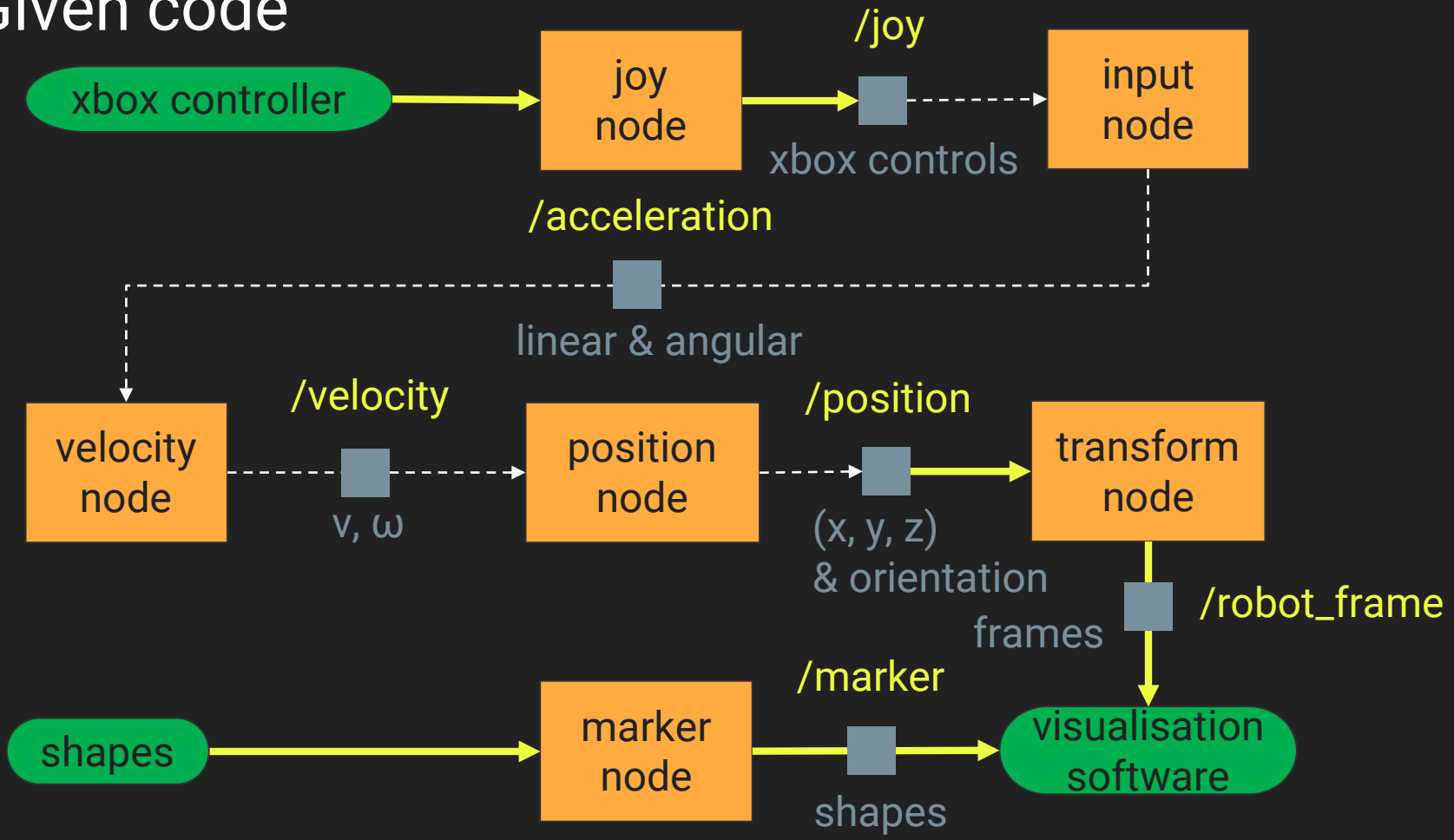
# Assignment 2 – End goal



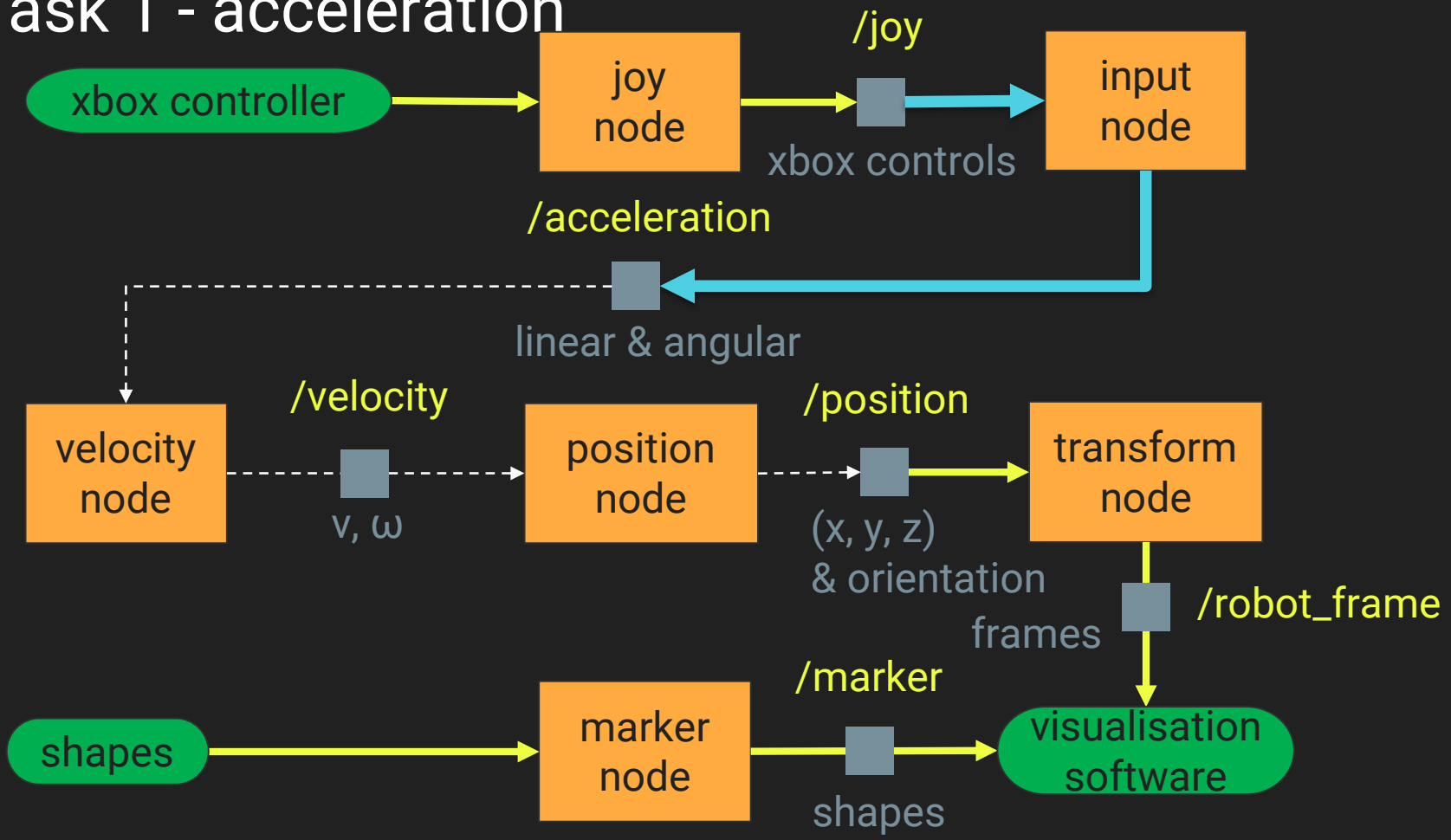
# Task Descriptions

- Task 1: joystick controls → accelerations
  - Implement the input node (subscribe to joy, publish accel)
- Task 2: accelerations → velocity → position
  - Implement the velocity node (subscribe to accel, publish velocity)
  - Implement the position node (subscribe to velocity, publish pose)
- Task 3: config file parsing
  - Read from a “config.txt” file
  - Parse data into useful structs

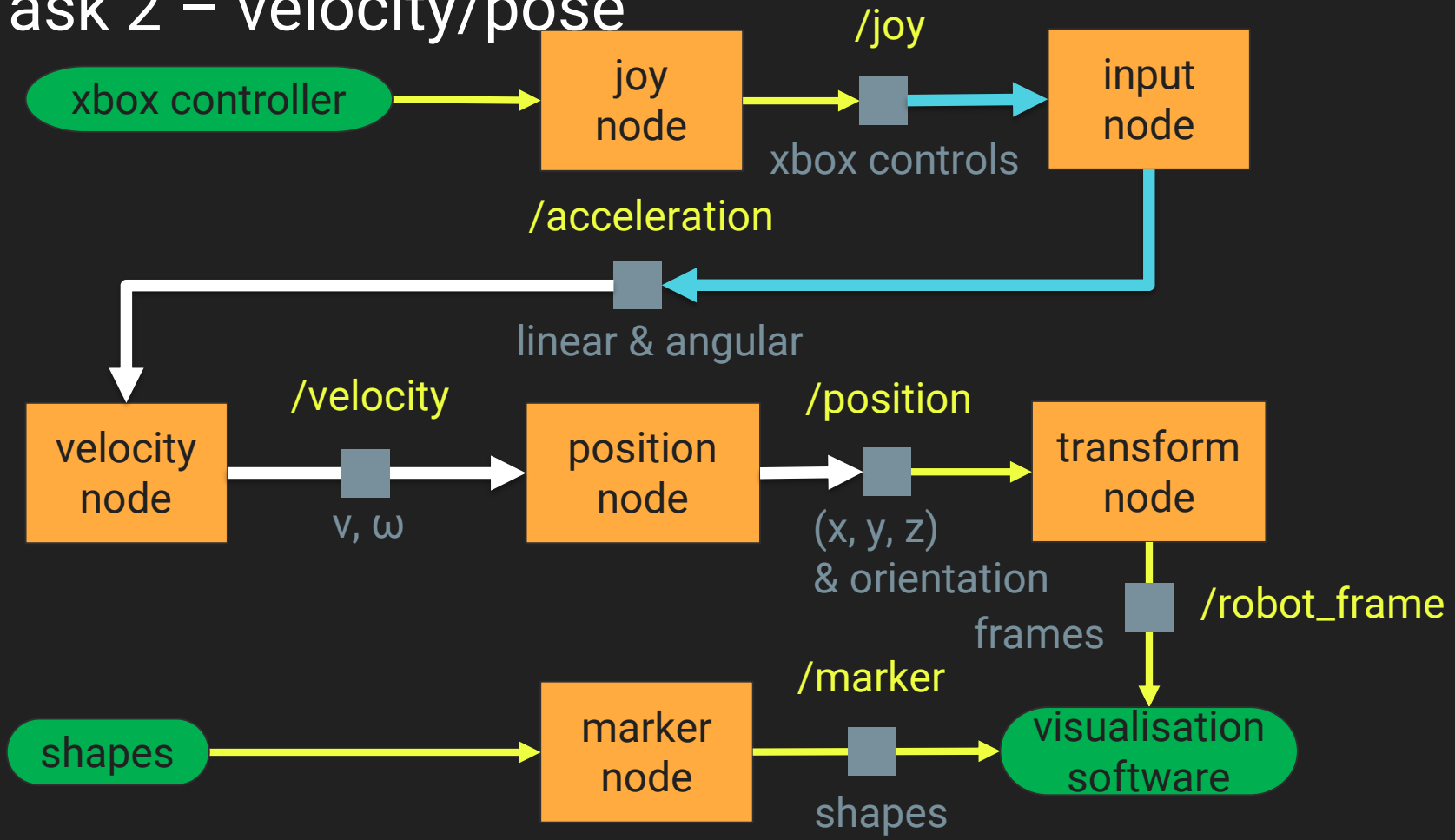
# Given code


xbox controller
joy node
/joy
xbox controls
input node
/acceleration
linear & angular
velocity node
/velocity
v, ω
position node
/position
(x, y, z) & orientation
transform node
/robot_frame
frames
shapes
marker node
/marker
shapes
visualisation software

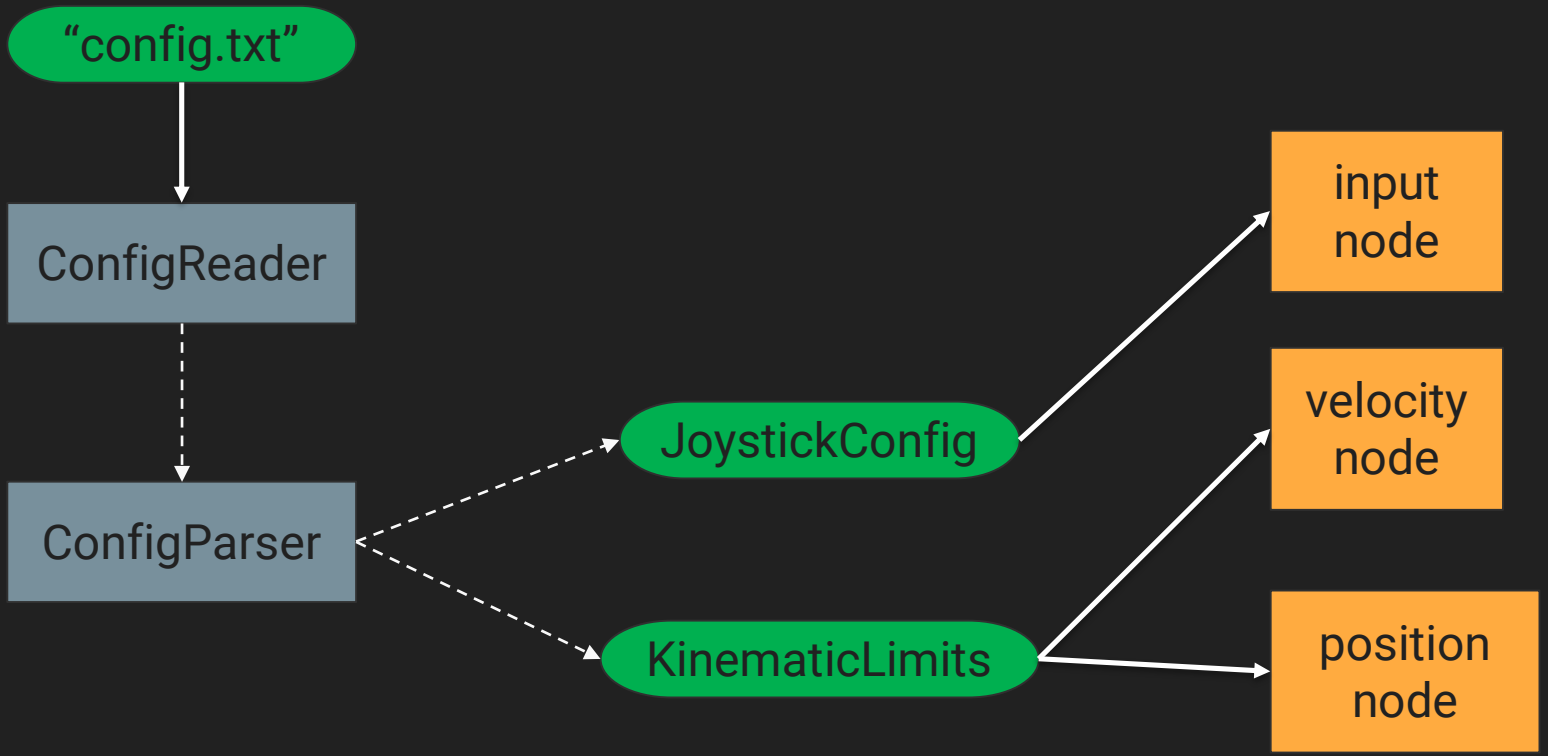
# Task 1 - acceleration



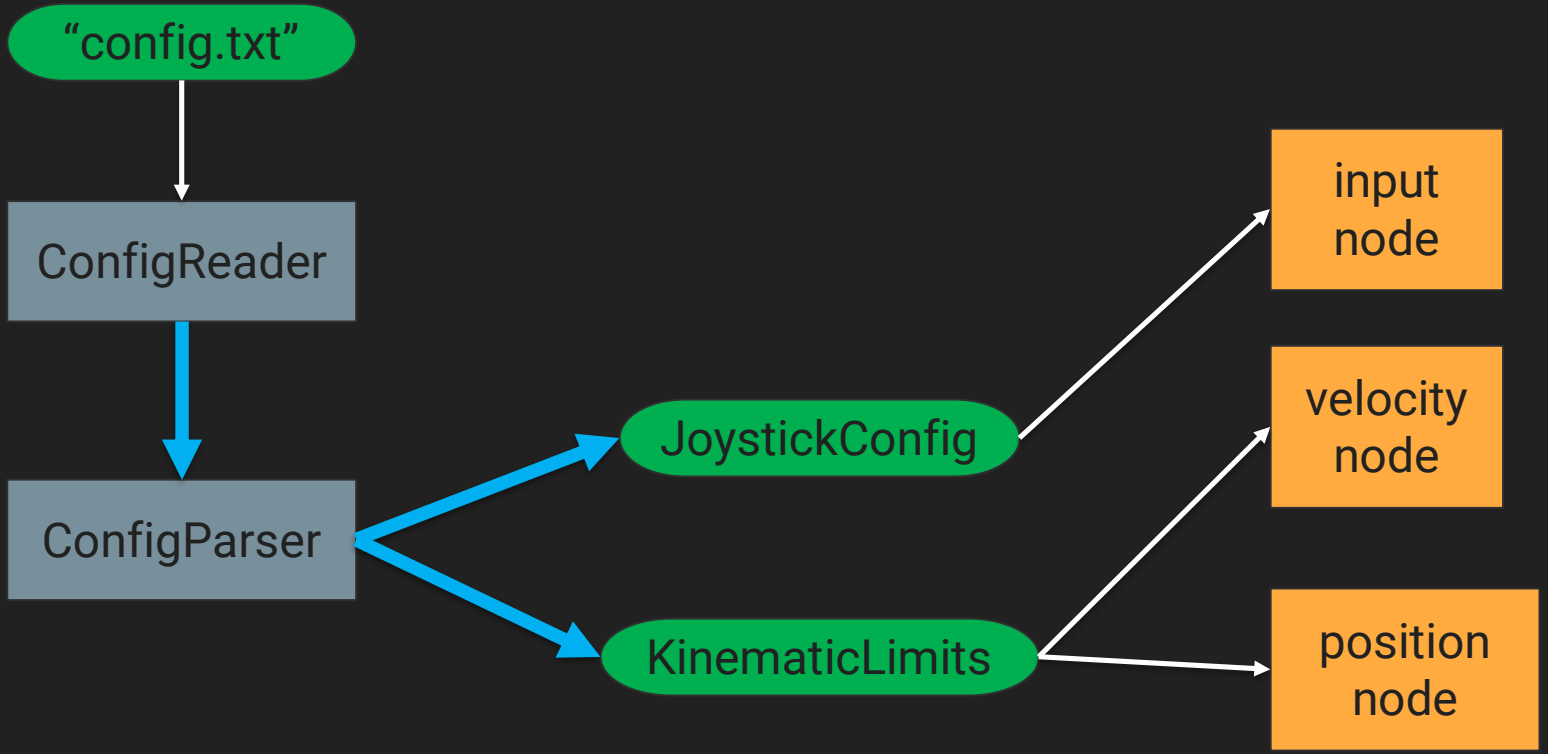
# Task 2 – velocity/pose



# Task 3 – given code



# Task 3 – config file parsing



# Nodes – are to be implemented using classes

| Node name | Class name |
|---|---|
| Input node | JoystickListener |
| Velocity node | VelocityKinematic |
| Position node | PoseKinematic |
| Transform node | TransformBroadcaster |
| Marker node | MarkerBroadcaster |

- Base code provided
- Fill in the skeleton classes

# ROS2 command line tools

- ros2 --help
- ros2 node list
- ros2 node info /some_node
- ros2 topic list
- ros2 topic info /some_topic
- ros2 topic echo /some_topic
- ros2 topic pub /some_topic msg/MessageType "data: value“

# ros2 --help

```
mtrn2500@mtrn2500-VirtualBox:~$ ros2 --help
usage: ros2 [-h] Call `ros2 <command> -h` for more detailed usage. ...

ros2 is an extensible command-line tool for ROS 2.

optional arguments:
  -h, --help            show this help message and exit

Commands:
  action     Various action related sub-commands
  component  Various component related sub-commands
  daemon     Various daemon related sub-commands
  launch     Run a launch file
  lifecycle  Various lifecycle related sub-commands
  msg        Various msg related sub-commands
  multicast  Various multicast related sub-commands
  node       Various node related sub-commands
  param      Various param related sub-commands
  pkg        Various package related sub-commands
  run        Run a package specific executable
  security   Various security related sub-commands
  service    Various service related sub-commands
  srv        Various srv related sub-commands
  topic      Various topic related sub-commands

  Call `ros2 <command> -h` for more detailed usage.
```

# ros2 node list

```
mtrn2500@mtrn2500-VirtualBox:~$ ros2 node --help
usage: ros2 node [-h]
                 Call `ros2 node <command> -h` for more detailed usage. ...

Various node related sub-commands

optional arguments:
  -h, --help            show this help message and exit

Commands:
  info  Output information about a node
  list  Output a list of available nodes

  Call `ros2 node <command> -h` for more detailed usage.
```

```
mtrn2500@mtrn2500-VirtualBox:~$ ros2 node list
/z0000000_input_node
/z0000000_velocity_node
/z0000000_pose_node
/z0000000_marker_node
/z0000000_transform_node
/launch_ros
/z0000000/joy_node
```

# ros2 node info /some_node

```
mtrn2500@mtrn2500-VirtualBox:~$ ros2 node info /z0000000/joy_node
/z0000000/joy_node
  Subscribers:
    /z0000000/parameter_events: rcl_interfaces/msg/ParameterEvent
  Publishers:
    /z0000000/joy: sensor_msgs/msg/Joy
    /z0000000/parameter_events: rcl_interfaces/msg/ParameterEvent
    /z0000000/rosout: rcl_interfaces/msg/Log
  Services:
    /z0000000/joy_node/describe_parameters: rcl_interfaces/srv/DescribeParameters
    /z0000000/joy_node/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
    /z0000000/joy_node/get_parameters: rcl_interfaces/srv/GetParameters
    /z0000000/joy_node/list_parameters: rcl_interfaces/srv/ListParameters
    /z0000000/joy_node/set_parameters: rcl_interfaces/srv/SetParameters
    /z0000000/joy_node/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomically
```

# ros2 topic list

```
mtrn2500@mtrn2500-VirtualBox:~$ ros2 topic --help
usage: ros2 topic [-h] [--include-hidden-topics]
                  Call `ros2 topic <command> -h` for more detailed usage. ...

Various topic related sub-commands

optional arguments:
  -h, --help            show this help message and exit
  --include-hidden-topics
                        Consider hidden topics as well

Commands:
  bw     Display bandwidth used by topic
  delay  Display delay of topic from timestamp in header
  echo   Output messages from a topic
  hz     Print the average publishing rate to screen
  info   Print information about a topic
  list   Output a list of available topics
  pub    Publish a message to a topic

  Call `ros2 topic <command> -h` for more detailed usage.
```

```
mtrn2500@mtrn2500-VirtualBox:~$ ros2 topic list
/parameter_events
/rosout
/tf
/z0000000/joy
/z0000000/marker
/z0000000/parameter_events
/z0000000/pose
/z0000000/rosout
```

# ros2 topic info /some_topic

```
mtrn2500@mtrn2500-VirtualBox:~$ ros2 topic info /z0000000/joy
Topic: /z0000000/joy
Publisher count: 1
Subscriber count: 0
```

# ros2 topic echo /some_topic

```
mtrn2500@mtrn2500-VirtualBox:~$ ros2 topic echo /z0000000/joy
header:
  stamp:
    sec: 1572575859
    nanosec: 665337311
  frame_id: joy
axes: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
buttons: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
---
header:
  stamp:
    sec: 1572575859
    nanosec: 752900206
  frame_id: joy
axes: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
buttons: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
---
```

# Q: Can I test without an xbox controller?
# A: Yes, you can! Use “ros2 topic pub” command

```
mtrn2500@mtrn2500-VirtualBox:~$ ros2 topic pub --help
usage: ros2 topic pub [-h] [-r N] [-p N] [-1] [-n NODE_NAME]
                      [--qos-profile {system_default,sensor_data,services_default,parameters,pa
rameter_events,action_status_default}]
                      [--qos-reliability {system_default,reliable,best_effort}]
                      [--qos-durability {system_default,transient_local,volatile}]
                      topic_name message_type [values]

Publish a message to a topic

positional arguments:
  topic_name            Name of the ROS topic to publish to (e.g. '/chatter')
  message_type          Type of the ROS message (e.g. 'std_msgs/String')
  values                Values to fill the message with in YAML format (e.g.
                        "data: Hello World"), otherwise the message will be
                        published with default values

optional arguments:
  -h, --help            show this help message and exit
  -r N, --rate N        Publishing rate in Hz (default: 1)
  -p N, --print N       Only print every N-th published message (default: 1)
  -1, --once            Publish one message and exit
  -n NODE_NAME, --node-name NODE_NAME
                        Name of the created publishing node
  --qos-profile {system_default,sensor_data,services_default,parameters,parameter_events,action
_status_default}
                        Quality of service profile to publish with
  --qos-reliability {system_default,reliable,best_effort}
                        Quality of service reliability setting to publish
                        with. (Will override reliability value of --qos-
                        profile option)
  --qos-durability {system_default,transient_local,volatile}
                        Quality of service durability setting to publish with.
                        (Will override durability value of --qos-profile
                        option)
```

# ros2 topic pub /some_topic msg/MessageType "data: value“

```
mtrn2500@mtrn2500-VirtualBox:~$ ros2 topic pub /z0000000/joy sensor_msgs/msg/Joy
 "header:
>   stamp:
>     sec: 1572437268
>     nanosec: 419506984
>   frame_id: joy
> axes: [-0.0, 0.0, 0.0, -0.0, -0.0, 0.0, 0.0, 0.0]
> buttons: [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]"
publisher: beginning loop
publishing #1: sensor_msgs.msg.Joy(header=std_msgs.msg.Header(stamp=builtin_inte
rfaces.msg.Time(sec=1572437268, nanosec=419506984), frame_id='joy'), axes=[-0.0,
 0.0, 0.0, -0.0, -0.0, 0.0, 0.0, 0.0], buttons=[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
)

publishing #2: sensor_msgs.msg.Joy(header=std_msgs.msg.Header(stamp=builtin_inte
rfaces.msg.Time(sec=1572437268, nanosec=419506984), frame_id='joy'), axes=[-0.0,
 0.0, 0.0, -0.0, -0.0, 0.0, 0.0, 0.0], buttons=[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
)
```

```
mtrn2500@mtrn2500-VirtualBox:~$ ros2 topic echo /z0000000/joy
header:
  stamp:
    sec: 1572437268
    nanosec: 419506984
  frame_id: joy
axes: [-0.0, 0.0, 0.0, -0.0, -0.0, 0.0, 0.0, 0.0]
buttons: [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
---
```