Tutorial on How to add new sensors to MaRS

This tutorial explains on how to add new sensors to MaRS framework and how to use the new sensor modules in ROS. This guide strictly follows the online MaRS tutorials.

Installation

This installs the MaRS ROS framework, which in turn installs the MaRS library. Please also refer to the online tutorial, which also provides a guide on how to use MaRS in a precompiled docker environment.

1. Create a new worspace root folder and `src` folder in it.
2. Initialize the workspace with `catkin init` (we typically use the `catkin_tools`, you might need to install them)
3. cd into the `src` folder and `git clone` the Mars ROS repository
4. cd into mars_ros and do a `git sumbodule update --recursive --init`. This will download also the corresponding Mars framework library.
5. cd into your workspace root and compile the workspace with `catkin build`

Adding a new Sensor

Library

To add a new sensor in the MaRS framework, first the sensor measurement functions and updates have to be implemented. For this example implementation a `pose` sensor is referenced with the measurement equation

$$z_p = \mathbf{P}_{WI} + \mathbf{R}_{WI}\ \mathbf{P}_{IP} \tag{1}$$

$$z_q = \mathbf{q}_{WI} \otimes \mathbf{q}_{IP} \tag{2}$$

To add the new sensor:

1. Create a new folder in `mars_ros/mars_lib/source/mars/include/mars/<SENSOR_NAME>`
2. Create 3 header files inside this folder (or copy them from e.g. the `pose` sensor)
   - `<SENSOR_NAME>_measurement_type.h`
   - `<SENSOR_NAME>_sensor_state_type.h`
   - `<SENSOR_NAME>_sensor_class.h`
3. (optional) for better usage, we typically also create the corresponding `.cpp` files and only have the definition in the `.h` file. This is up to the user however.
4. `<SENSOR_NAME>_measurement_type.h` defines the data type class of the measurement. E.g. for the `pose` sensor it includes
   - position: `Eigen::Vector3d position_`
   - attitude: `Eigen::Quaternion<double> orientation_;`

5. `<SENSOR_NAME>_sensor_state_type.h` defines additional (auxiliary) states, that a sensor needs to update the core states used in the EKF. In the case of the `pose` sensors these are the calibration states between the inertial frame I and the pose sensor frame P, i.e.
   - `p_ip_` translational calibration state
   - `q_ip_` orientational calibration state
6. `<SENSOR_NAME>_sensor_class.h` defines on how the core states and auxiliary states are initialized, updated, and residuals/Jacobians are calculated. Taking the `pose` sensor as an example:
   1.
   2. Write the initialization of the additional sensor states in `BufferDataType Initialize(...)`
   3. In `bool CalcUpdate(...)` depending on the sensor measurement model implement the
      1. Measurement noise matrix calculation
      2. Implementation of the measurement jacobian H
      3. Residual calculation
   4. In `<SENSOR_NAME>StateType ApplyCorrection(...)` implement the auxiliary sensor state corrections depending on the measurement model.
7. Finally add the new files to the `mars_ros/mars_lib/source/mars/CMakeLists.txt` under the `set(headers ...)` path. Additionally if you have added `.cpp` files put them also into the `set(sources ...)` declaration

If you perform these steps then the sensor can be used as a generalized object in the MaRS framework.

ROS

To add the newly created sensor in the ROS-based MaRS framework

1. Create (or copy) a new wrapper class in `mars_ros/include/mars_wrapper_<SENSOR_NAME).h` and the corresponding `.cpp` file in `mars_ros/src/mars_wrapper_<SENSOR_NAME).cpp`
2. Copy the `ImuMeasurementCallback` from any of the other sensors
3. Add ROS callback functions for all the sensors used. Each function should perform the following:
   1. Derive the timestamp
   2. Convert the ROS message to the MaRS measruement type
   3. Perform the by calling the `ProcessMeasurement` function of the `core_logic`
4. Copy or add the `RunCoreStatePublisher()` method.

Finally, also add the new files to the `mars_ros/CMakeLists.txt`, e.g. for the pose sensor:

```
add_executable(pose_node src/mars_node.cpp src/mars_wrapper_pose.cpp)
add_dependencies(pose_node ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS} ma
target_link_libraries(pose_node
```

```
  ${catkin_LIBRARIES} ${MARS_LIBRARIE}
)
set_property(TARGET pose_node PROPERTY COMPILE_DEFINITIONS POSE)
```

Also update the `mars_ros/src/mars_node.cpp` to include the new wrapper.