CMU - Robotics Institute **Manipulation Lab - Simple Hands Project**

Logger node - ROS - Introduction

<manipulationLab@gmail.edu> November 2, 2011

1 Logger

Logger_node is a semiautomated solution for logging the state of the system for specific periods of time. Logger_node provides functionality to log to file all topics published to the Mlab ROS network, append information to log files and manage the location in the computer of those same files. The default folder where log files are being saved is *log*, relative to the global log folder defined by the system parameter *logger/folder*.

Logger_node currently samples and logs, when available, the following topics:

- robot_CartesianLog
- robot_JointsLog
- robot_ForceLog
- hand_EncodersLog
- hand_AnglesLog

In the future it should also automatically log, when present, the topics published by the Mocap and Kinect nodes. The rate at which logger samples all topics is specified by the system parameter *logger/sampligRate*.

Besides logging to file, logger_node publishes an aggregated topic with all relevant information of the current state of the system. The frequency is also specified by *logger/samplingRate*. It reflects the exact same information being logged to file. For this reason, it is convenient to use this topic for data driven/learning applications. the aggregated log contains two timers. The first one reflects the global ROS time. The second one is always zero, except if logging to file, when reflects the time since the logging procedure stated. The second one can be used to synchronize real time date with previously logged data.

It is composed of two ROS packages: logger_node, a standalone typical ROS package that provides topics and services, and logger_comm, including the message definitions and a C++ client class to simplify the invocation and communication with logger_node.

1.1 logger_node

(Located at `svnroot/code/nodes/logger/ROS/logger_node`)

Services:

- **logger_Start**: Service to start logging to file.

```
int64 id         # Experiment id (added to the first line of the log file).
string folder    # Folder where to save it (relative to logFolder/log)
---
string filename  # Complete name of the logFile (relative to the logFolder)
int64 ret        # Set to 1 (success) or 0 (error)
string msg       # Error description
```

- **logger_Stop**: Service to stop logging and close the file.

```
---
int64 ret        # Set to 1 (success) or 0 (error)
string msg       # Error description
```

- **logger_Create**: Service to create an empty log file.

```
---
string filename    # Complete name of the logFile (relative to the logFolder)
int64 ret          # Set to 1 (success) or 0 (error)
string msg         # Error description
```

- **logger_Append**: Service to append a string to the end of a log file.

```
string filename    # Name of the logFile where to append information (relative to logFolde
string info        # String to append.
---
int64 ret          # Set to 1 (success) or 0 (error)
string msg         # Error description
```

- **logger_Copy**: Service to copy a log file into a different folder. The specified folder is always relative to the main logFolder. This service can be used to copy files of a specific experiment into a dedicated folder.

```
string filename      # Name of the logFile to copy (realtive to the logFolder).
string folder        # Folder where to copy (relative to the logFolder)
---
int64 ret            # Set to 1 (success) or 0 (error)
string msg           # Error description
```

Topics:

- **logger_SystemLog**: Topic that aggregates all other topics published to the ROS network.

```
float64 timeStamp    # ROS time-stamp
float64 logtimeStamp # Time since the beginning of last logStart.
int64 id             # Id of file begin logged.
float64 x            # x-coordinate of the robot.
float64 y            # y-coordinate of the robot.
float64 z            # z-coordinate of the robot.
float64 q0       # q0-coordinate of the robot.
float64 qx           # qx-coordinate of the robot.
float64 qy      # qy-coordinate of the robot.
float64 qz      # qz-coordinate of the robot.
float64 j1      # joint 1 of the robot.
float64 j2      # joint 2 of the robot.
float64 j3      # joint 3 of the robot.
float64 j4      # joint 4 of the robot.
float64 j5      # joint 5 of the robot.
float64 j6      # joint 6 of the robot.
float64 fx      # force in the x direction.
float64 fy      # force in the y direction.
float64 fz      # force in the z direction.
float64 tx      # torque in the x direction.
float64 ty      # torque in the y direction.
float64 tz      # torque in the z direction.
int64 encMotor      # motor encoder.
int64[] encFinger    # array of finger encoders.
float64 angleMotor   # motor angle.
float64[] angle      # array of finger angles.
```

1.2 logger_comm

(Located at svnroot/code/nodes/logger/ROS/logger_comm)
Logger_comm is a ROS wrapped C++ class meant to simplify the communication with logger_node. It contains:

1. Message and Service definitions. When creating a ROS application that uses logger_node, the application only needs to be dependent on logger_comm. This avoids having to link and compile against the entire logger_node.

2. C++ client class LoggerComm that handles all configuration and initialization required to use logger_node. As an example, if we want to call the service logger_Start, we only need to instantiate a LoggerComm object and call to the member routine loggerStart. Otherwise we would have to include all required header files, subscribe to the logger_Start service and format the required message to be sent to the service.

Usage example:

```
...
ros::NodeHandle node;
```

```
  LoggerComm logger;           //Create LoggerComm client.
  logger.subscribe(&node);     //Subscribe to all services of logger_node.
  logger.Start();              //Start logging to file.
  ...
  logger.Stop();               //Stop logging to file.
  ...
```

Member routines:

```
class LoggerComm
{
  public:
    LoggerComm();
    LoggerComm(ros::NodeHandle* np);
    ~LoggerComm();

    // Subscribe Function
    void subscribe(ros::NodeHandle* np);

    // Subscribe to Topics
    void subscribeSystem(ros::NodeHandle* np, int q_len,
        void (*funcPtr)(const logger_comm::logger_SystemLogConstPtr&));

    // Shutdown service clients
    void shutdown();

    bool Start(string &filename, string folder, int id=0);
    bool Start(string &filename, int id=0);
    bool StartGrasp(string &filename, int id=0);
    bool StartPlace(string &filename, int id=0);
    bool Start(int id=0);
    bool StartGrasp(int id=0);
    bool StartPlace(int id=0);
    bool Stop();
    bool Append(string filename, string info);
    bool Create(string &filename);
    bool Copy(string filename, string folder);
...
};
```