# Information for Students at the Robotics and Perception Group

January 6, 2014

## Contents

# 1 General Information for the Project

## 1.1 Start of the Project

- Fill out a form *FirstName_LastName.docx* with your personal data and send it to your supervisor.
- You have to hand in a project schedule to your supervisors after the first two weeks.
- You will dedicate your first two weeks to studying papers and learning about ROS[1], Git[2] and Ubuntu (if necessary). Furthermore, you will be given an initial set of papers to read and additional material by your supervisors.

## 1.2 During the Project

- Your supervisor will schedule a weekly meeting where you will have to tell him what you did in the previous week and what problems you are having. Also, in this meeting, you will agree together on what to the next week.
- If you have questions you can ask your supervisor anytime, not only at the meeting.

## 1.3 Presentation

- The presentations will last 20 minutes each and will be followed by 10 minutes of questions and 5 minutes of internal discussion only among the lab members.
- Please train your presentation, make sure it works with the beamer and make sure you do not go beyond these 20 minutes.
- You can find the presentation template in the *Templates* folder.
- You can find the evaluation template we use in the *Templates* folder.

## 1.4 End of the Project

- Clean up your code, document it and push it on *GitHub*.
- Hand in a DVD containing:
  1. PDF of your report with all Latex sources
  2. Your Presentation (as ppt and pdf) and Videos
  3. All the data you used to make the plots and run the experiments
  4. All the code and documentations up to date
- You will receive your grade as soon as **all** the previous points are accomplished!

[1] `http://www.ros.org`
[2] `http://www.github.com`

# 2 Notation

## 2.1 Variable styles in LaTeX

Use lowercase and bold letters for vectors, e.g. $\mathbf{x}$, uppercase and bold letters for matrices, e.g. $\mathbf{R}$, and lowercase letters with normal weight for scalars, e.g. $s$.

## 2.2 Coordinate Systems and Rotations

We use the notation introduced by Glocker in the course "Mechanik 3" at ETHZ to express coordinate frames, rotations and vectors. Refer to Chapter 5 "Kinematik" in the lecture skript for more details [3]. Figure 1 gives an overview of how coordinate transformations and vectors are specified. Observe that the coordinate system in which a vector is expressed is always written as index before the variable, e.g. ${}_B\mathbf{t}_{AB}$ is the vector from $A$ to $B$ expressed the coordinate system $B$. For the ease of reading, the index for the origin coordinate frame can be omitted: ${}_O\mathbf{t}_k := \mathbf{t}_k$.

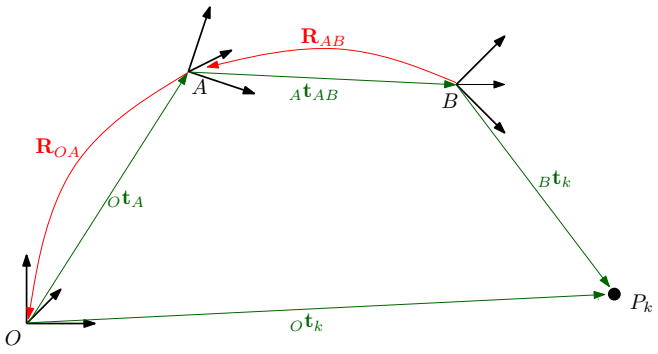


Figure 1: Notation overview.

$A$ and $B$ are two adjacent coordinate frames and $O$ is the frame of origin. $\mathbf{R}_{AB}$ describes the coordinate transformation from frame $B$ to frame $A$, thus it holds that

$$
\begin{aligned}
{}_O\mathbf{t}_k &= \mathbf{R}_{OB}\ {}_B\mathbf{f}_k \\
\mathbf{R}_{OB} &= \mathbf{R}_{OA}\ \mathbf{R}_{AB}
\end{aligned}
$$

Always match the variable names in the code with the variable names in the documentation. Use the following notation in latex and in your `source code` respectively:

$$
\begin{aligned}
{}_B\mathbf{t}_k &= \texttt{t\_k\_in\_b} \\
\mathbf{R}_{OB} &= \texttt{R\_o\_b}
\end{aligned}
$$

For ease of reading it is even better if you write out the name of the frames as follows in your code: `R_o_b` = `R_origin_body`.

[3] `http://mitschriften.amiv.ethz.ch/main.php?page=3&scrid=1&pid=87&eid=1`

## 2.3 Measured, estimated and target values

For controllers and estimators please specify the variables as follows in the report and the source code respectively:

| | | |
|---|---|---|
| true value: | $\mathbf{x}$ | `x` |
| estimated value: | $\hat{\mathbf{x}}$ | `x_estimated` |
| measured value: | $\tilde{\mathbf{x}}$ | `x_measured` |
| desired value: | $\mathbf{x}_{\text{des}}$ | `x_desired` |
| error value: | $\mathbf{x}_{\text{e}}$ | `x_error` |
| equilibrium value: | $\mathbf{x}^*$ | `x_equilibrium` |

# 3 Programming Guidelines

## 3.1 Using Git

We use *GitHub* to store our code. You will have to create an account on *GitHub*[4] and tell your user name to you supervisor. He will then give you access to our repositories. You will only be granted pull access to repositories with a *rpg_∗* which you have to *fork* first in order to get your own copy of the repository and being able to make changes.

We also maintain a Wiki on *GitHub*[5] where you can find more technical information which are useful for your project.

## 3.2 C++ Style guide

Follow the ROS style guide[6]. Most important follow these rules:

- Variables are `under_scored`. Choose descriptive instead of cryptic names! For rotations and transformations you can use capital letters `R` and `T`.
- Functions are `camelCased`.
- Files are `under_scored` and have the extension `.cpp` or `.h`
- Classes are `CamelCased`.
- Namespaces are `under_scored`.
- If something is not specified in the ROS guidelines, consider the Google C++ style[7]
- Put `.cpp` files in the `src/` folder and `.h` files in `include/package_name/`.
- SPACES only! One tab is 2 spaces.
- Be consistent

Give a brief description above each function and class in the header files. Use the Doxygen styles[8].

ROS provides a style-file for Eclipse which you can download [9].

The following two listings provide an example of the style.

[4] `http://www.github.com`
[5] `https://github.com/ailab/rpg_tools/wiki`
[6] `http://www.ros.org/wiki/CppStyleGuide`
[7] `http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml`
[8] `http://www.stack.nl/~dimitri/doxygen/manual/docblocks.html`
[9] `http://www.ros.org/wiki/IDEs#Auto_Formatting`

Listing 1: point.h

```
namespace project_name {

/// Class description here
class Point
{
private:
  double x_;    //!< x coordinate of the point
  double y_;    //!< y coordinate of the point

public:
  Point(double x, double y);
  ~Point();

  /// Compute distance to other point
  double getDistance(const Point& other) const;
};

} // end namespace project_name
```

Listing 2: point.cpp

```
#include <project_name/point.h>
#include <math.h>

namespace project_name {

Point::Point(double x, double y) :
  x_(x),
  y_(y)
{}

Point::~Point()
{}

double Point::getDistance(const Point& other) const
{
  double dx = x_ - other.x_;
  double dy = y_ - other.y_;
  return sqrt(dx * dx + dy * dy);
}

} // end namespace project_name
```