Odometry Calculation of the Tricycle-Drive

Nov. 15, 2016

Junpyo Hong (jp7.hong@gmail.com)

Table of Contents

1. Tricycle-Drive

The presented mobile platform has a tricycle-drive configuration. Tricycle drive employs a single driven front wheel and two passive rear wheels and it is fairly common in AGV applications because of their inherent simplicity[1]. Tricycle-drive cannot control the heading angle θ independently, and it means that we cannot compute changes in θ.

Generally, the overall degree of freedom that a mobile platform can manipulate, called the *degree of maneuverability*, δ_M can be readily defined in terms of mobility and steerability as follows[2]:

$$\delta_M = \delta_m + \delta_s \tag{1}$$

where δ_m is *degree of mobility* and δ_s is *degree of steerability*.

In case of the tricycle WMR, degree of maneuverability is as follows:

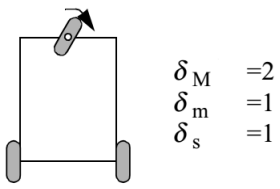


Fig. 1. Degrees of maneuverability, mobility, and steerability of the tri-cycle drive

This configuration allows the ICR (Instantaneous Center of Rotation) to range anywhere on the plane and ICR lie on a predefined line with respect to the mobile platform reference frame. In a tricycle, this line extends from the shared common axle of the rear fixed wheels, with the steerable front wheel setting the ICR point along this line. The ICR lies on the line that passes through, and is perpendicular to the fixed rear wheels.

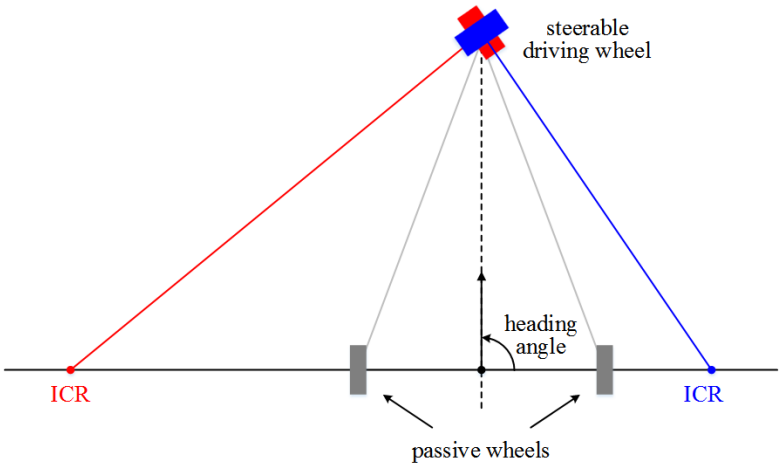


Fig. 2. ICR (Instantaneous Center of Rotation) of the tricycle drive

The control variables of the mobile platform consists of steering direction $\alpha(t)$ and angular velocity of the steering wheel $\omega(t)$. As particular cases are as follows:

$$\alpha(t) = 0°, \quad \omega(t) = \omega \quad \rightarrow \text{moves straight} \tag{2}$$

$$\alpha(t) = 90°, \quad \omega(t) = \omega \quad \rightarrow \text{rotates in place} \tag{3}$$

Fig. 3 shows case of Eq. (3) and ICR coincides with the special point of the mobile platform and the mobile platform rotates in place[3].

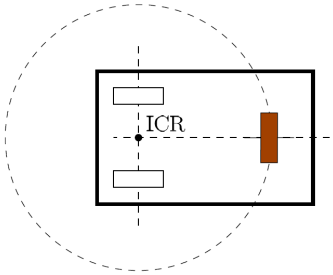


Fig. 3. Case that steering direction $\alpha(t) = 90°$

Direct kinematics of the tricycle-drive can be derived as follows[4]:

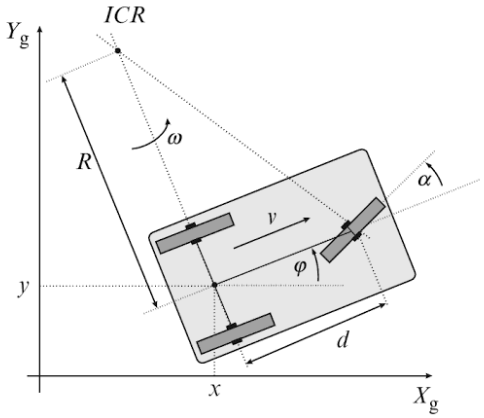


Fig. 4. Tricycle-drive kinematics

Steering wheel radius, r

Linear velocity of the steering wheel, $v_s(t) = \omega_s \cdot r$

Distance between the mobile platform center and ICR, $R(t) = d \cdot tan(\frac{\pi}{2} - \alpha(t))$ (4)

Angular velocity of the moving frame, $\omega(t) = \frac{\omega_s(t) \cdot r}{\sqrt{d^2}} \sin \alpha(t)$

In the world frame, tricycle drive mechanism has a kinematics model as follow[5]:

$$\dot{x} = v_s(t)\, cos\big(\alpha(t)\big)\, cos\big(\theta(t)\big)$$

$$\dot{y} = v_s(t) \cos\big(\alpha(t)\big) \sin\big(\theta(t)\big) \quad (5)$$

$$\dot{\theta} = \frac{v_s(t)}{d} \sin(\alpha(t))$$

where $v = v_s(t) \cos(\alpha(t))$, $\omega = \frac{v_s(t)}{d} \sin(\alpha(t))$, and v_s is the rim velocity of the steering wheel.

In our system, we can use gyro's estimated heading angle. Therefore, we can substitute gyro angle for $\dot{\theta}$ of Eq. (5). We called it *gyro-assisted odometry*[6].

2. Brief Description of Folders and Files

```
├──doc: Document file
├──doxygen: Doxygen working directory
├──prj: Project directory
│    ├──ubuntu: Ubuntu makefile
│    └──vs2013: Visual Studio 2013 solution and project file
│          └──gnuplot: Program to show the simulation result
│                └──platforms
└──src: Source files
```

3. Source Files

Filename	Description
main.cpp	main() function
math2.h	Miscellaneous mathematical definitions and functions
pGNUPlot.h	C++ interface class declaration (open source)
pGNUPlot.cpp	C++ interface class implementation (open source)
Pose.h	Struct definition of Position (x, y) and Pose (x, y, heading_angle)
Singleton.h	Singleton template class
stdafx.h	Header file for VS2013 pre-compilation
stdafx.cpp	Source file for VS2013 pre-compilation
targetver.h	Header file for VS2013 pre-compilation
TestTricycle.h	Test class declaration for the simulation of *Tricycle* class
TestTricycle.cpp	Test class implementation for the simulation of *Tricycle* class
Tricycle.h	Odometry calculation class declaration for tricycle-drive
Tricycle.cpp	Odometry calculation class implementation for tricycle-drive
VirtualGyro.h	Virtual gyro class declaration
VirtualGyro.cpp	Virtual gyro class implementation

4. Odometry Calculation Routine

```
///
/// @brief      pose estimator interface member function
///
/// @param      time [in] time of reading of the input data (unit: sec)
/// @param      steering_angle [in] steering wheel angle (unit: rad)
/// @param      encoder_ticks [in] number of ticks from the traction motor
///             encoder (unit: ticks (integer))
/// @param      angular_velocity [in] reading from a gyroscope measuring the
///             rotation velocity of the platform around the Z axis
///             (unit: rad/s)
///
/// @return     new estimated pose. Tuple (x, y, heading) representing the
///             estimated pose of the platform (unit: m, m, rad)
///
SPose CTricycle::Estimate(float time, float steering_angle, int encoder_ticks, \
    float angular_velocity)
{
    // Front wheel radius = 0.2 m
    // Back wheels radius = 0.2 m
    // Distance from front wheel to back axis (r) = 1m
    // Distance between rear wheels (d) = 0.75m
    // Front wheel encoder = 512 ticks per revolution

    // circumference of a wheel:
    //     2 * M_PI * 0.2 = 0.4 * M_PI = 1.2566370614359172953850573533118f (m)
    // distance per tick:
    //     (0.4 * M_PI) / 512 = 0.0024543692606170259675489401431 9 (m/pulse)

    /// previous timestamp (sec)
    static float fPrevTime = 0.f;

    /// time difference since previous time
    float fDiffTime = time - fPrevTime;

    /// distance of the front steering wheel
    float fFrontWheelDist = encoder_ticks * m_fFrontDistPerTick;

    /// front wheel velocity (m/s)
    //@{
    float fFrontWheelVel = 0.f;
    if (!almostZero(fDiffTime)) ///< prevent divide by zero
        fFrontWheelVel = fFrontWheelDist / fDiffTime;
    //@}

    /// angle difference (rad)
    float fDiffQ = (fFrontWheelVel / m_fDistBtwFrontRear) * sinf(steering_angle);
```

```
    /// get the angular velocity from gyro (rad/s)
    float fW = CVirtualGyro::GetInstance()->GetAngVel();

    /// consider time difference
    m_pose.q += fW * fDiffTime;

    /// clamp angle
    m_pose.q = AngleClamp(m_pose.q);

    /// differences of robot position (x, y)
    float fDiffX = (fFrontWheelVel * fDiffTime) * cosf(steering_angle) \
       * cosf(m_pose.q);
    float fDiffY = (fFrontWheelVel * fDiffTime) * cosf(steering_angle) \
       * sinf(m_pose.q);

    /// update the robot pose
    m_pose.x += fDiffX;
    m_pose.y += fDiffY;

    /// update timestamp for the next time
    fPrevTime = time;

    /// return robot pose (x, y, heading)
    return m_pose;
}

///
/// @brief      Pose estimator interface function for the Tricycle mobile robot
///
/// @param      time [in] time of reading of the input data (unit: sec)
/// @param      steering_angle [in] steering wheel angle (unit: rad)
/// @param      encoder_ticks [in] number of ticks from the traction motor
///             encoder (unit: ticks (integer))
/// @param      angular_velocity [in] reading from a gyroscope measuring the
///             rotation velocity of the platform around the Z axis
///             (unit: rad/s)
///
/// @return     new estimated pose. Tuple (x, y, heading) representing the
///             estimated pose of the platform (unit: m, m, rad)
///
SPose estimate(float time, float steering_angle, int encoder_ticks, \
    float angular_velocity)
{
    /// return calculated odometry pose (x, y, heading)
    return CTricycle::GetInstance()->Estimate(time, steering_angle, \
        encoder_ticks, angular_velocity);
}
```

5. Simulation Results

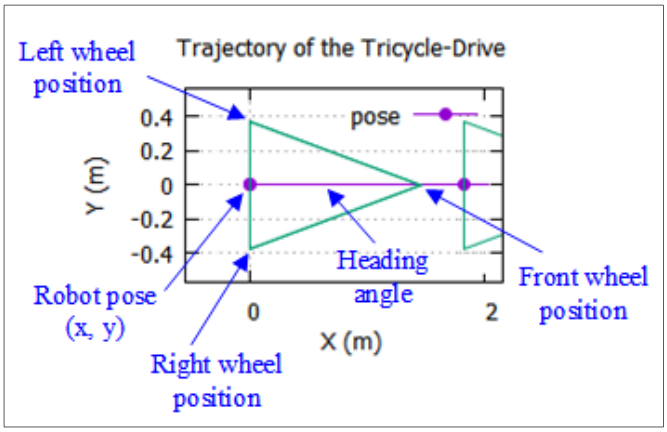
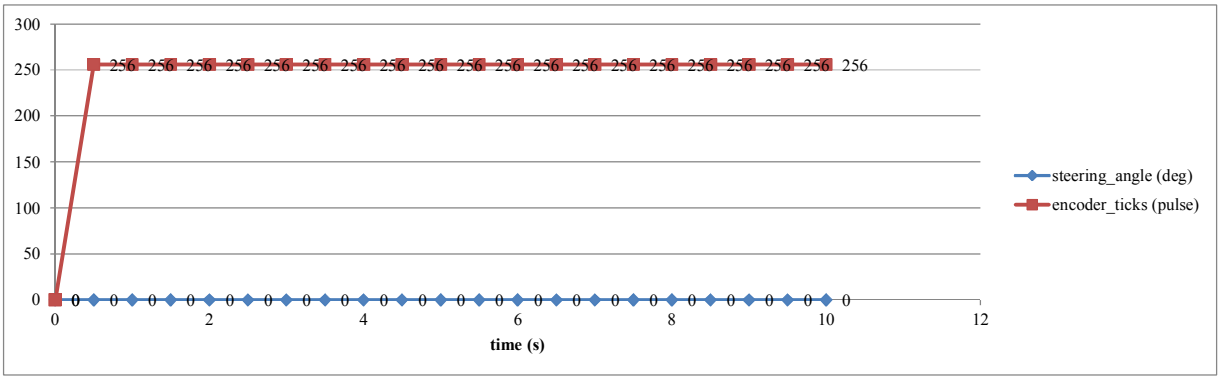


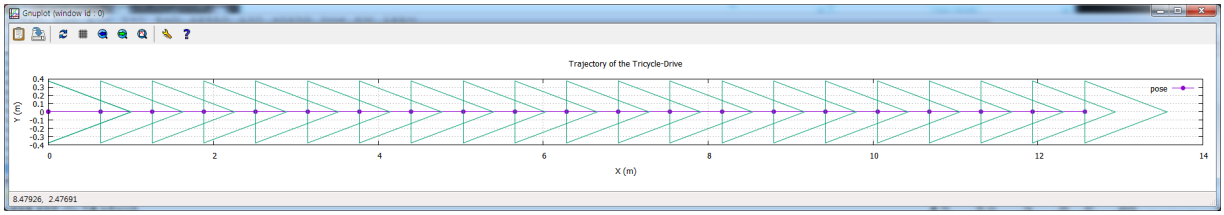
Fig. 5. Meaning of simulation plot

Test Case #1 – Move Straight

Input: steering_angle = 0, encoder_ticks = 256 ticks / 0.5s (maximum speed) for 10 seconds.

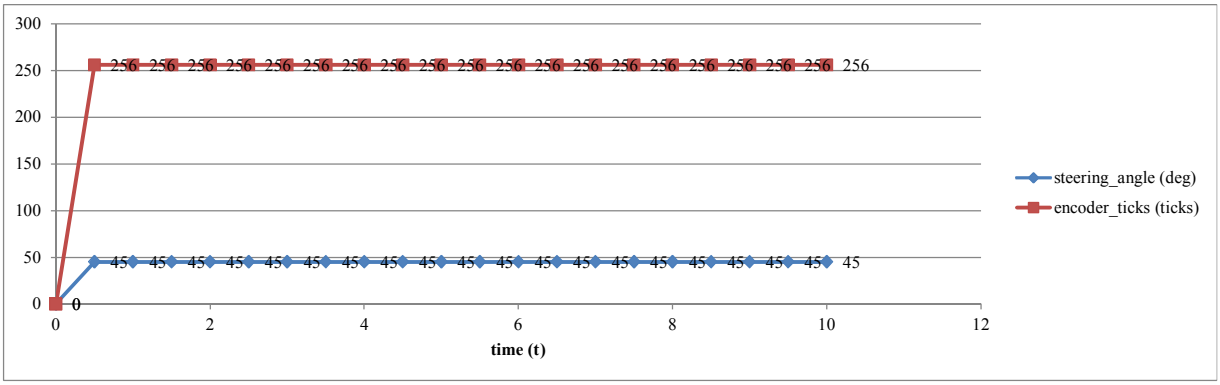


Result: The mobile platform moves 12.566372m correctly. (2 * PI * 0.2m (front wheel radius) * 10 seconds)

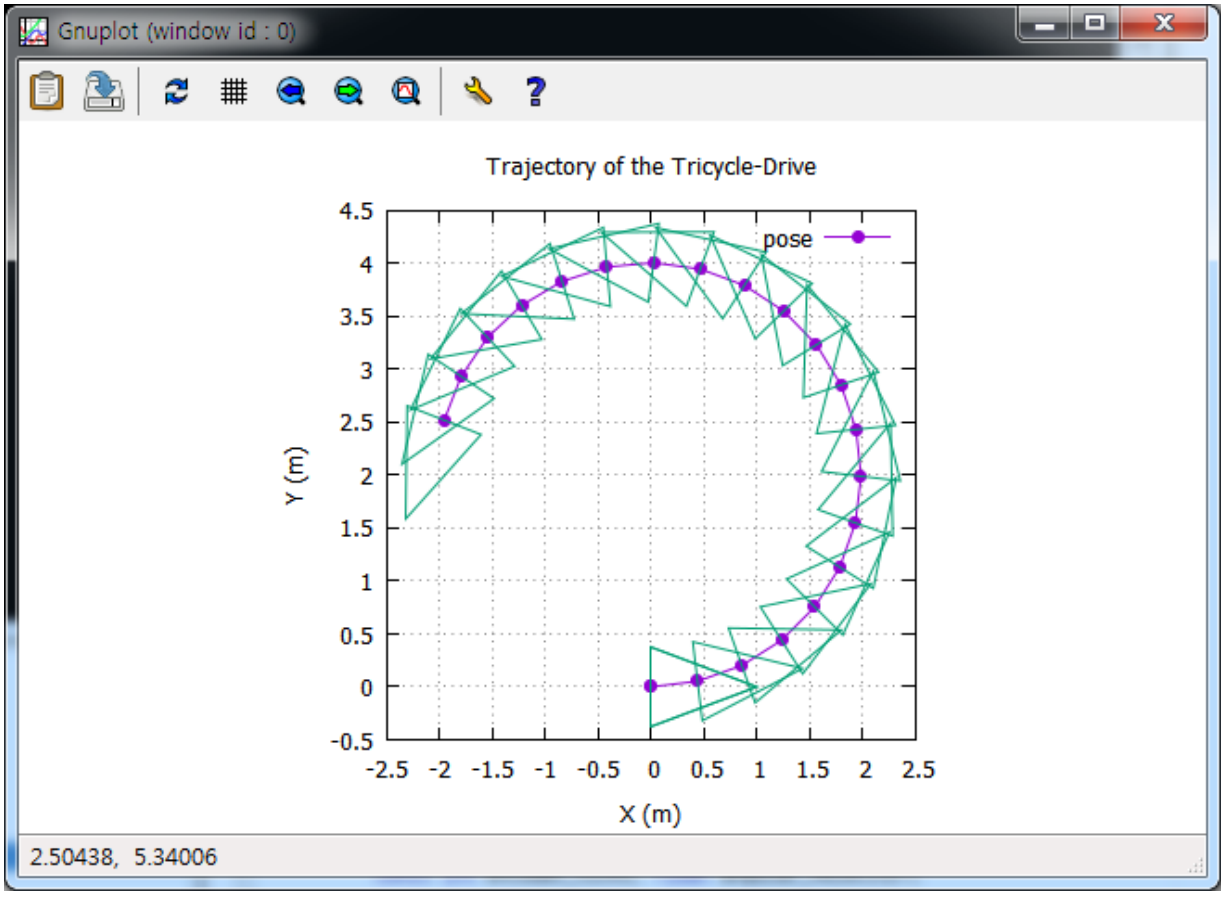


Test Case #2 – Turn

Input: steering_angle = 45°, encoder_ticks = 256 ticks / 0.5s (maximum speed) for 10 seconds.

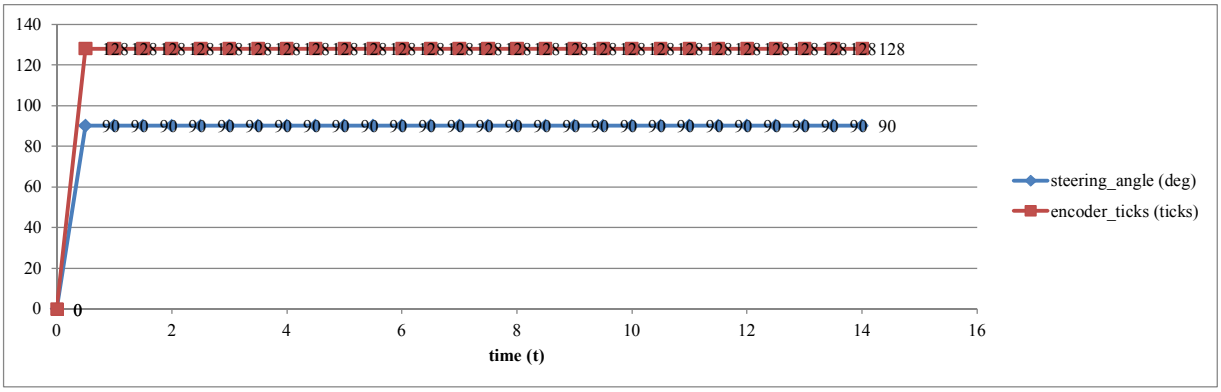


Result: The mobile platform revolves on ICR (Instantaneous Center of Rotation).

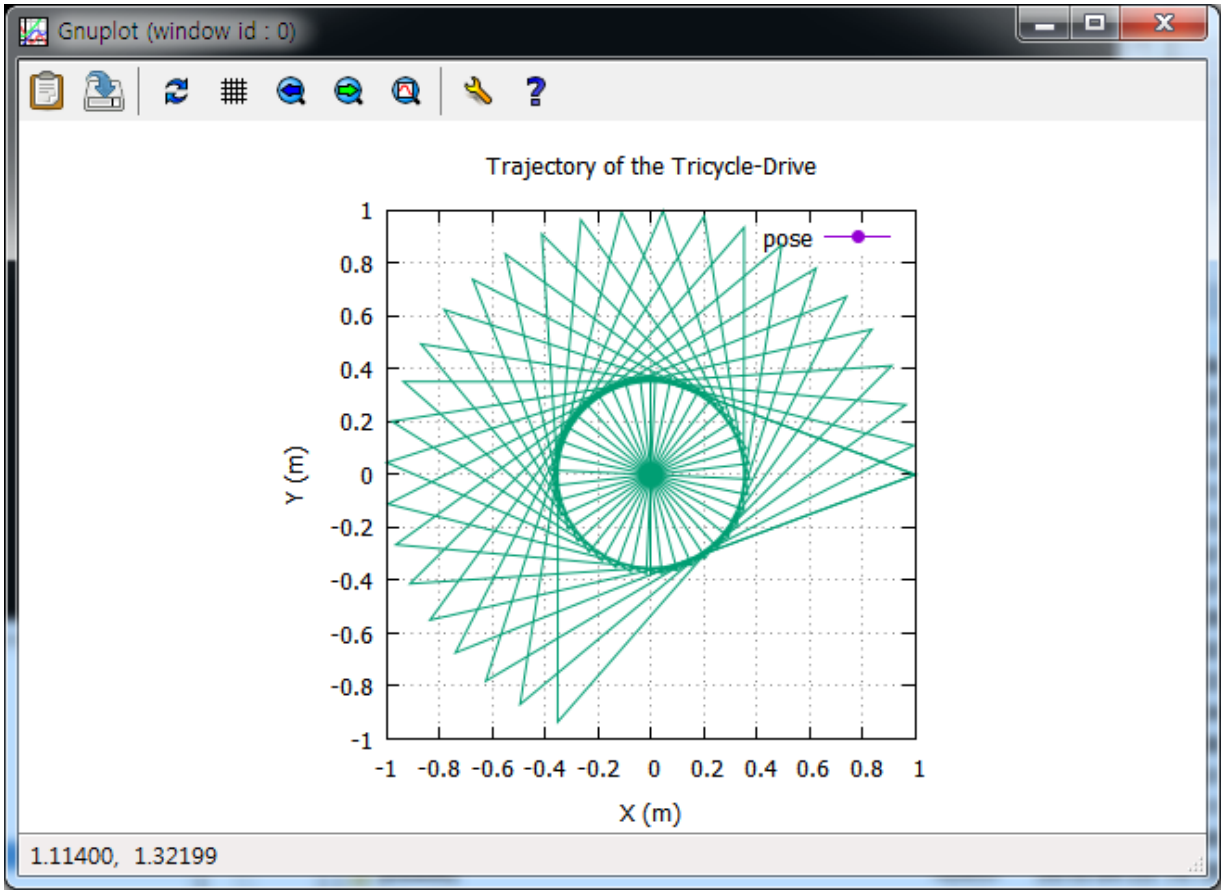


Test Case #3 – Rotate In Place

Input: steering_angle = 90°, encoder_ticks = 128 ticks / 0.5s (maximum speed) for 14 seconds.

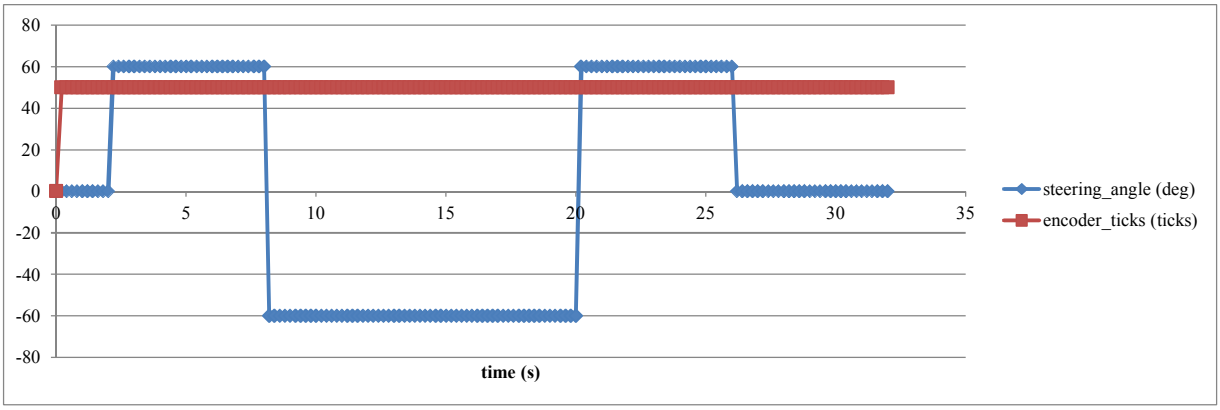


Result: The mobile platform rotates in place.

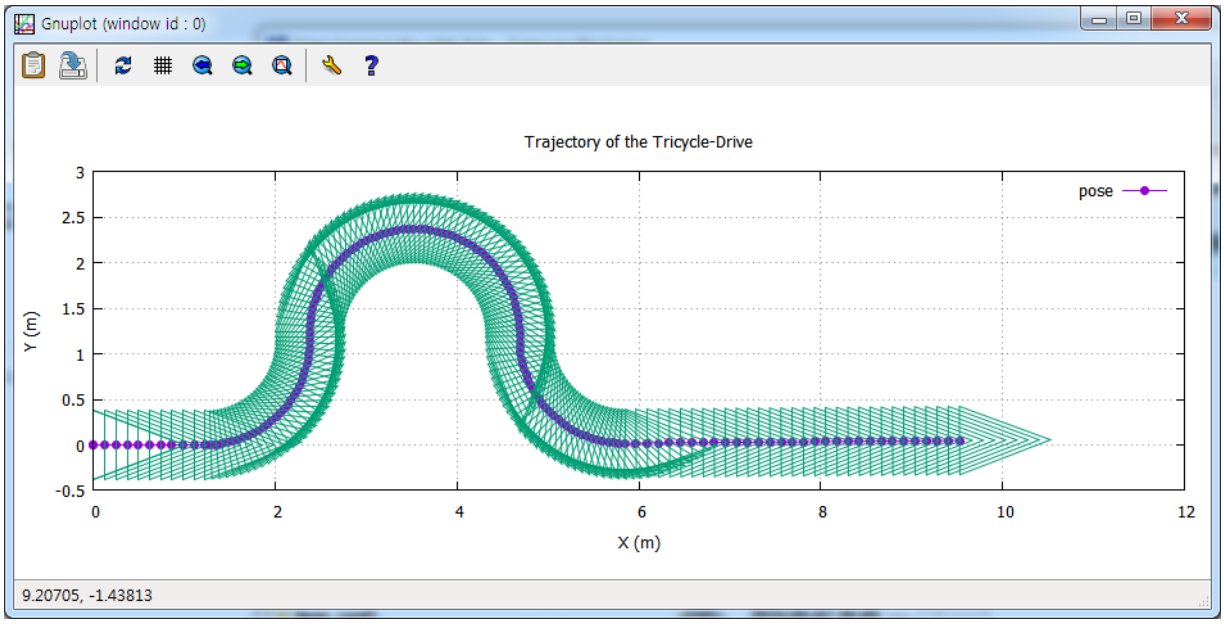


Test Case #4 – Mix

Input: steering_angle = 0 or 60°, encoder_ticks = 50 ticks / 0.2s (almost full speed) for 32 seconds.

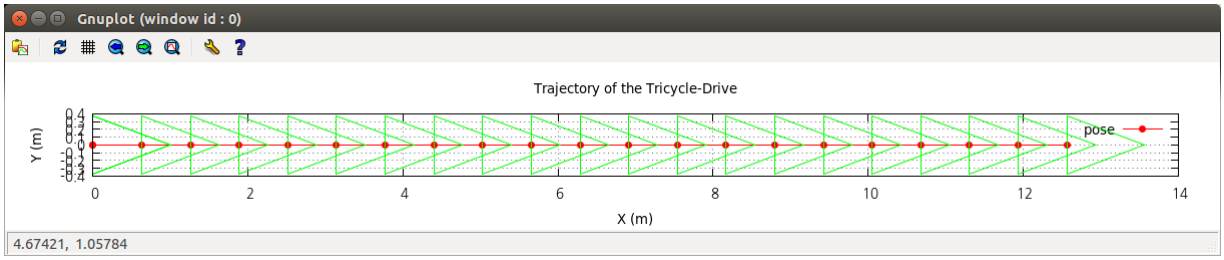


Result: The mobile platform rotates in place.

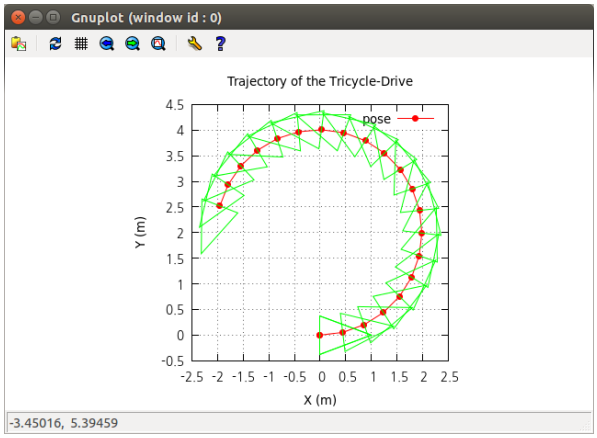


Test Case #1~#4 – Ubuntu

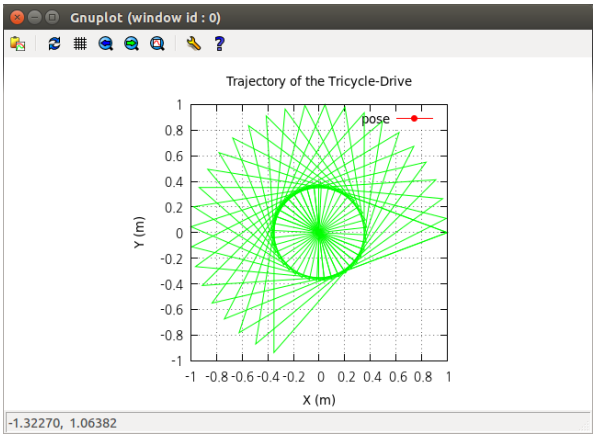
Test Case #1 on Ubuntu



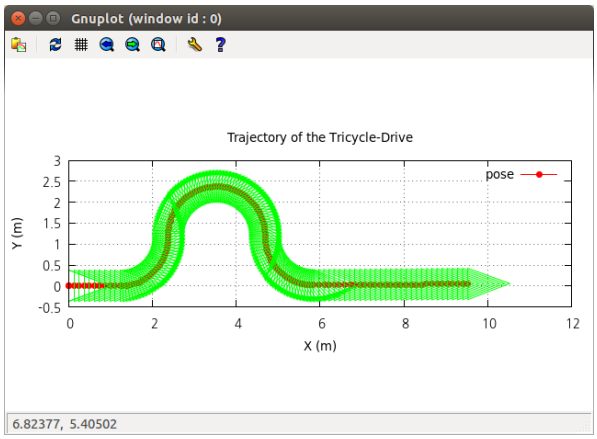
Test Case #2 on Ubuntu



Test Case #3 on Ubuntu



Test Case #4 on Ubuntu



Appendix

- Where to download Visual Studio 2013 Express (.iso file):

  http://download.microsoft.com/download/2/5/5/255DCCB6-F364-4ED8-9758-EF0734CA86B8/vs2013.3_dskexp_ENU.iso

- How to make *Visual Studio 2013 solution file* and *ubuntu Makefile* (Prerequisite installation: CMake):
  - Visual Studio 2013 (Please execute following batch file while VS2013 is not running.)

    ...\Tricycle\prj\vs2013>make_sln.bat
  - Ubuntu

    .../Tricycle/prj/ubuntu$ ./makefile.sh

- How to clean *Visual Studio 2013 solution file* and *ubuntu Makefile*:
  - Visual Studio 2013 (Please execute following batch file while VS2013 is not running.)

    ...\Tricycle\prj\vs2013>clean.bat
  - Ubuntu

    .../Tricycle/prj/ubuntu$ ./clean.sh

- How to build:
  - Visual Studio 2013
    - Open ...\Tricycle\prj\vs2013\Tricycle.sln solution file.
    - In the Solution Explorer in VS2013, set *Tricycle* project as StartUp project.
    - Press F7 to build the solution.
  - Ubuntu

    .../Tricycle/prj/ubuntu$ make

- How to run the execution file
  - Visual Studio 2013

    Run one of following built execution files.

    ...\Tricycle\prj\vs2013\Debug\Tricycle.exe

    ...\Tricycle\prj\vs2013\MinSizeRel\Tricycle.exe

    ...\Tricycle\prj\vs2013\Release\Tricycle.exe

    ...\Tricycle\prj\vs2013\RelWithDebInfo\Tricycle.exe
  - Ubuntu

    .../Tricycle/prj/ubuntu$ ./Tricycle

References

1. J. Borenstein, H. R. Everett, and L. Feng, *Where am I? Sensors and methods for mobile robot positioning*, University of Michigan, 1996
2. Roland Siegwart, Illah Reza Nourbakhsh and Davide Scaramuzza, *Introduction to Autonomous Mobile Robots*, MIT Press, Feb. 2011
3. http://msl.cs.uiuc.edu/mobile/mch2.pdf
4. http://chrome.ws.dei.polimi.it/images/7/75/Robotics_03_Mobile_Robots_Kinematics.pdf
5. Gregor Klancar, Andrej Zdesar, Saso Blazic, Igor Skrjanc, *Wheeled Mobile Robotics: From Fundamentals Towards Autonomous Systems*, Butterworth-Heinemann, Jan. 2017
6. Edwin Olson, "A Primer on Odometry and Motor Control," Dec. 2004

- End of Document -