Odometry

Wego & Industrial Robot

WeGo

목차

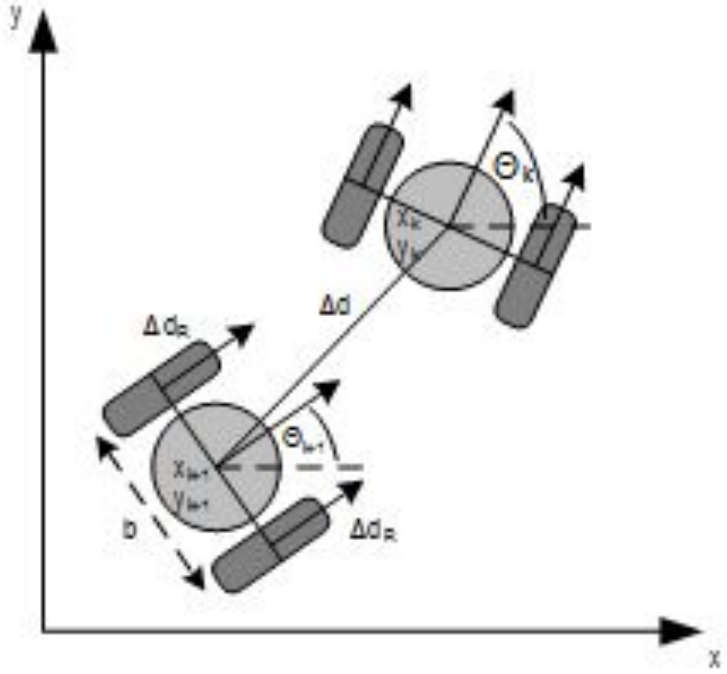
01

Odometry

WeGo

01 Odometry

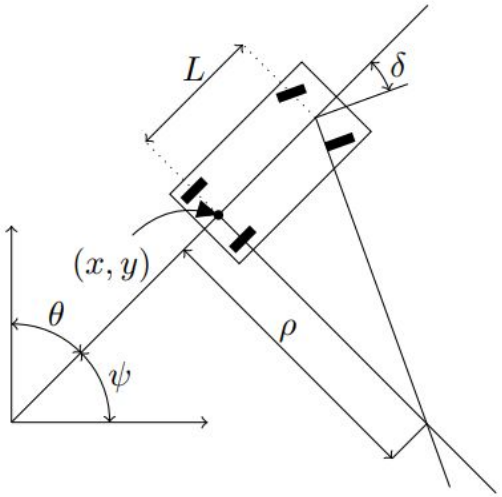
- Odometry는 말 그대로 주행기록계라는 의미
- 바퀴의 회전수(Encoder), IMU(관성 측정 장치) 등을 이용
- 위의 센서를 기반으로 움직이는 물체의 위치를 측정하는 방법



01 Odometry

- Odometry를 계산하기 위해서는 바퀴의 회전을 통해 이동을 계산하는 Model이 필요하며, Mobile Robot의 경우, Ackermann Steering Model과 Differential Drive Model의 두 가지가 대표적

- Ackermann Steering Model
- 일반적인 후방 구동, 전방 조향의 차량 모델
- 전방 조향을 통해 원 궤적을 그리는 형태
- 역으로 계산 시, 원하는 위치에 이동할 때, 어느 정도의 조향각이 필요한지 계산할 수 있음(Pure Pursuit)

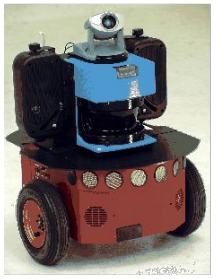


$$\dot{x} = v\cos(\psi) = v\cos\left(\frac{\pi}{2} - \theta\right)$$
$$\dot{y} = v\sin(\psi) = v\sin\left(\frac{\pi}{2} - \theta\right)$$
$$\dot{\theta} = \frac{v}{L}\tan(\delta).$$

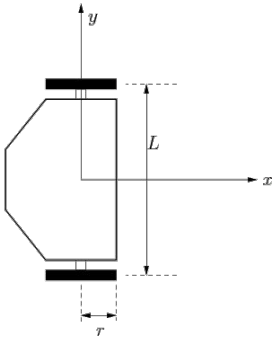
- Differential Drive Model
- 각 바퀴의 제어가 가능하며, 바퀴 사이의 속도 차이를 이용하여, 전, 후진 및 회전을 제어할 수 있는 형태
- 마찬가지로 역으로 계산 시, 원하는 위치에 이동할 때, 어느 정도의 조향각이 필요한지 계산할 수 있음(Pure Pursuit)

$$\dot{x} = \frac{r}{2}(v_l + v_r)cos(\theta)$$
$$\dot{y} = \frac{r}{2}(v_l + v_r)sin(\theta)$$
$$\dot{\theta} = \frac{r}{L}(v_r - v_l)$$

$$\dot{x} = vcos(\phi)$$
$$\dot{y} = vsin(\phi)$$
$$\dot{\phi} = \omega$$



(a)



(b)

01 Odometry

- Dead Reckoning
- Odometry와 유사하지만, 실외에서 GPS가 연결이 끊기거나 하는 경우, 차량 Model 및 차량 속도, Heading Angle 등의 정보를 이용하여, 차량의 위치를 추정하는 방법
- 역할 자체는 동일하며, 목적에 따라 용어가 달라짐

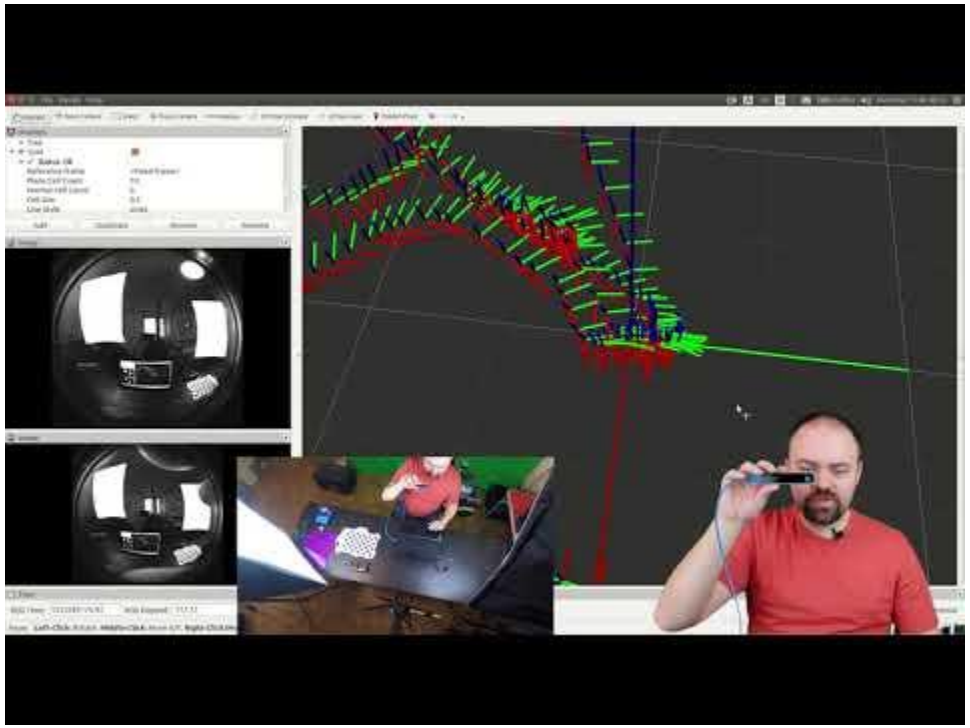
01 Odometry

- Visual Odometry
- Odometry를 Encoder 및 IMU 등 실제 이동에 해당하는 정보를 이용하여 진행하는 것이 아닌, 카메라 영상의 Frame과 Frame 차이를 비교하는 방식을 통해 Odometry를 계산하는 기술
- 카메라를 이용하여 6-DOF의 궤적을 계산
- 이동속도가 너무 빠르거나, Frame Drop 등이 발생할 때, 정확도가 급격히 감소
- Wheel Odometry와 IMU Odometry, Visual Odometry를 모두 퓨전해서 사용하면 정확도 향상에 큰 도움이 됨

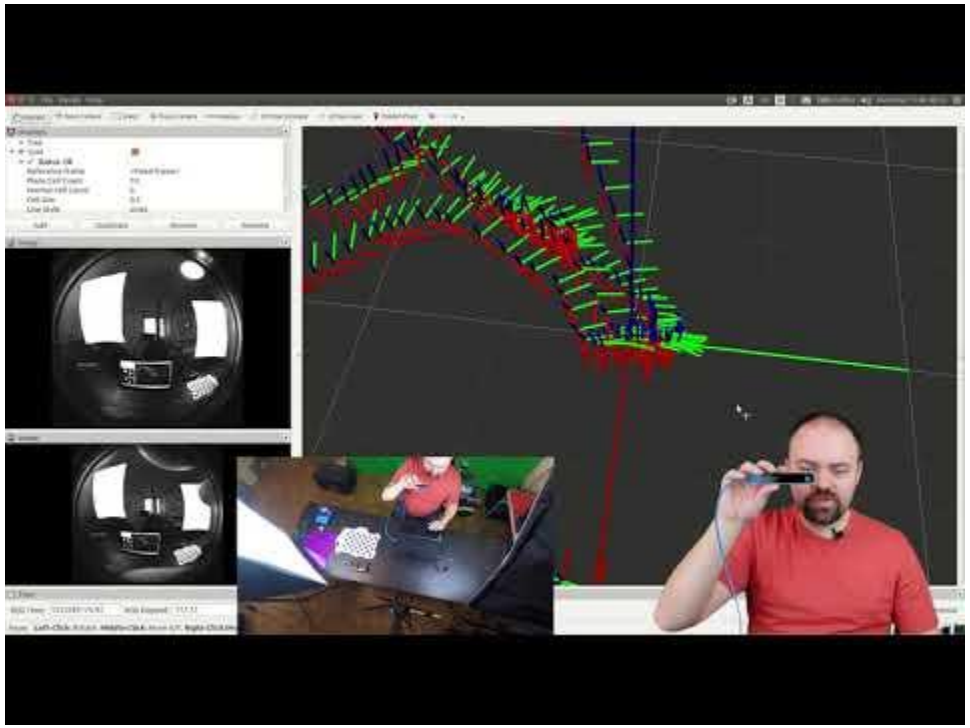
WeGo

01 Odometry

- T265 Tracking Camera
- Intel RealSense 계열이며, 내부적으로 Visual Odometry 기능 및 Visual SLAM 기능이 포함되어 있는 카메라



- T265 Tracking Camera
- Intel RealSense 계열이며, 내부적으로 Visual Odometry 기능 및 Visual SLAM 기능이 포함되어 있는 카메라



01 Odometry

- Scout mini Simulator Odometry Publisher
- ScoutStatus Message Type의 /scout_status를 Subscribe하여, Odometry를 Publish
- Odometry의 경우, nav_msgs/Odometry Message Type을 이용하여 Publish

01 Odometry

- scout_msgs/ScoutStatus
- std_msgs/Header header
    - uint32 seq
    - time stamp
    - string frame_id
  - float64 linear_velocity
  - float64 angular_velocity
  - float64 transverse_linear_velocity
  - uint8 base_state
  - uin8 control_mode
  - uint16 fault_code
  - float64 battery_voltage
  - scout_msgs/ScoutMotorState[4] motor_states
    - float64 current
    - float64 rpm
    - float64 temperature
  - bool light_control_enabled
  - scout_msgs/ScoutLightState front_light_state
    - uint8 mode
    - uint8 custom_value

WeGo

01 Odometry

- nav_msgs/Odometry

- std_msgs/Header header
  - uint32 seq
  - time stamp
  - string frame_id
- string child_frame_id
- geometry_msgs/PoseWithCovariance pose
  - geometry_msgs/Pose pose
    - geometry_msgs/Point position
      - float64 x
      - float64 y
      - float64 z
    - geometry_msgs/Quaternion orientation
      - float64 x
      - float64 y
      - float64 z
      - float64 w
  - float64 [36] covariance
- geometry_msgs/TwistWithCovariance twist
  - geometry_msgs/Twist twist
    - geometry_msgs/Vector3 linear
      - float64 x
      - float64 y
      - float64 z
    - geometry_msgs/Vector3 angular
      - float64 x
      - float64 y
      - float64 z
  - float64[36] covariance

01 Odometry

- rostopic echo /scout_status
- 상황에 맞춰서 Topic의 내용을 Subscribe해서 사용
- 가장 쉬운 방법은 생성되어있는 Linear Velocity 및 Angular Velocity의 값을 이용하여, 적분하는 방법을 통해 Odometry를 생성하는 방법 적용
- Linear Velocity의 경우 m/s의 단위를 사용
- Angular Velocity의 경우 rad/s의 단위를 사용
- 위 두 가지의 값을 Sampling Time과 곱하는 방법을 통해 이동한 거리 및 회전한 각도를 계산할 수 있음 → Odometry

01 Odometry

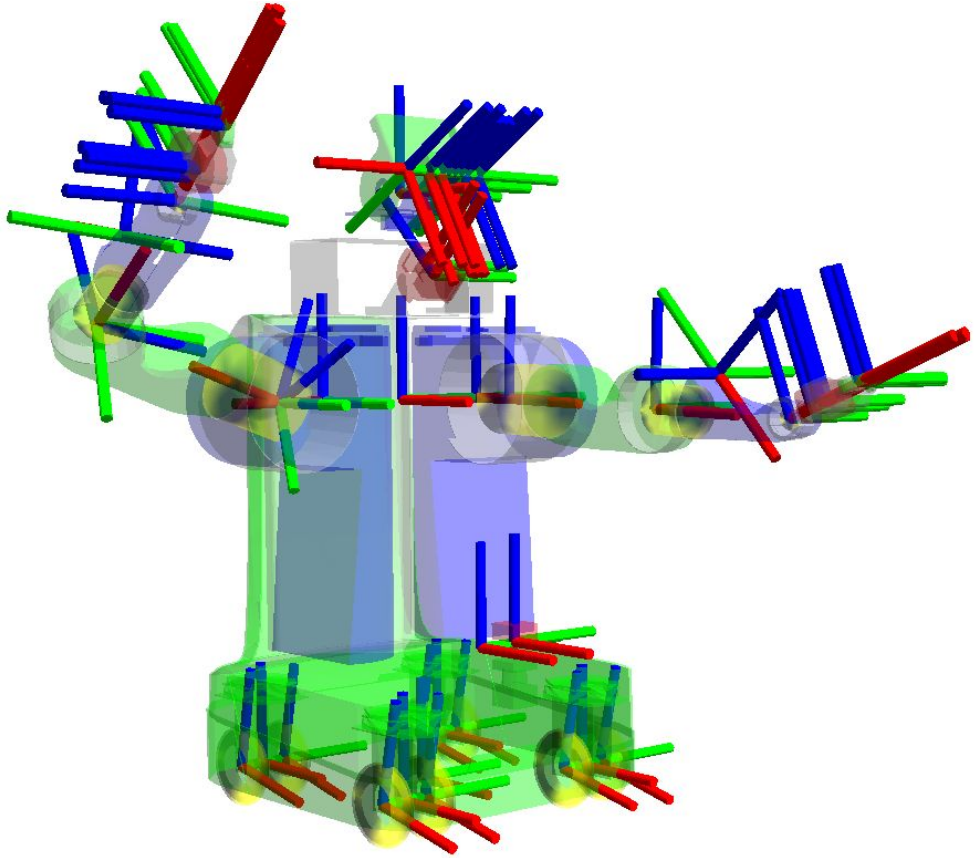
- 실제 Platform 구동을 통해서, Odometry 데이터를 검증
- Rviz 상의 TF들의 변화량과 실제 로봇의 이동량을 비교하는 방법을 통해 Odometry 데이터 오차의 누적을 확인합니다.
-
- Simulator의 Odom에 오차를 추가하여, 어느정도의 오차일 때, SLAM을 통해서 보정이 불가능해지는지 확인합니다.
- 위의 결과를 기반으로, 어느정도의 오차가 성능이 충분한지 확인합니다.

02

Transformation

- **Transformation**

- **TF는 Frame 사이의 변환 관계를 나타내는 Topic**



- **Transformation**
- **3D - 3D** 사이의 좌표 변환 관계를 나타내므로, 실제 필요한 값은 **6-DOF (x, y, z, roll, pitch, yaw)**의 **6**가지
- 변환 관계를 모두 정리하면 **Tree**의 형태로 확인할 수 있다.
- 일반적으로 특정 **Reference Frame**을 기준으로 **Sensor Data**값이 어떻게 들어오는지 변환하여 사용하거나, **Odometry** 정보를 나타낼 때 사용한다.

- **Transformation**
- **TF Demo**
- **$ sudo apt install ros-melodic-turtle-tf2 ros-melodic-tf2-tools ros-melodic-tf**
- **$ roslaunch turtle_tf2 turtle_tf2_demo.launch**
- http://wiki.ros.org/tf2
- https://github.com/ros-industrial-consortium/descartes_tutorials/issues/14
- https://github.com/ros/geometry_tutorials\
-

Q & A

go.support@wego-robotics.com

go.sales@wego-robotics.com