FUNCTIONAL and TECHNICAL REQUIREMENTS DOCUMENT

SymForce: Komputasi Simbolik dan Pembuatan Kode untuk Robotika

Nama : Muhammad Dani Ilham Alfafa Hakim
NIM :1103194036

Developed by:

1.0 GENERAL INFORMATION

1.1 Purpose

Tujuan dari dokumen ini adalah untuk mempelajari informasi mengenai simulasi robotika sebagai tugas dari mata kuliah Robotika dan Sistem Cerdas. Simulasi robot yang akan dibahas pada dokumen ini adalah simulasi robot Symforce yang dibuat oleh Symforce ORG.

1.2 Scope

Cakupan dari dokumen ini adalah untuk mensimulasikan simulasi robotika yang tersedia pada Symforce. Simulasi robot yang digunakan adalah simulasi robot pendeteksi arah gerak.

1.3 Project References

Dokumen-dokumen kunci yang mendukung dokumen ini tercantum di bawah ini sebagai referensi:

- Laman web Symforce : https://symforce.org/

1.4 Acronyms and/or Definitions

IDE	Lingkungan Pengembangan Terpadu yang menyediakan antarmuka pengguna untuk pengembangan kode, pengujian, dan fitur debug. Ini membantu mengatur artefak proyek yang relevan dengan kode sumber aplikasi perangkat lunak.
Python	bahasa pemrograman tingkat tinggi untuk keperluan umum. Filosofi desainnya menekankan keterbacaan kode dengan penggunaan lekukan yang signifikan.
C++	bahasa pemrograman untuk membangun perangkat lunak. Ini adalah bahasa berorientasi objek yang berfokus pada objek (bidang data yang memiliki atribut unik) daripada logika atau fungsi.
Pypi	Python Package Index (PyPI) adalah gudang perangkat lunak untuk bahasa pemrograman Python.
Apache	Apache adalah perangkat lunak server web sumber terbuka dan gratis yang menggerakkan sekitar 40% situs web di seluruh dunia. Nama resminya adalah Apache HTTP Server, dan dikelola serta dikembangkan oleh Apache Software Foundation. Ini memungkinkan pemilik situs web untuk menyajikan konten di web — oleh karena itu dinamai “server web”.

2.0 CURRENT SYSTEM SUMMARY

SymForce adalah komputasi simbolik dan pembuatan kode perpustakaan yang menggabungkan kecepatan pengembangan dan fleksibilitas matematika simbolik di Python dengan kinerja kode yang dibuat secara otomatis dan sangat dioptimalkan dalam C++ atau target apa pun bahasa waktu proses. SymForce memungkinkan untuk membuat kode masalah sekali dengan Python, bereksperimen dengannya secara simbolis, menghasilkan kode yang dioptimalkan, lalu jalankan pengoptimalan yang sangat efisien masalah berdasarkan definisi masalah asli. Pendekatan Symforce dimotivasi oleh pengembangan algoritma untuk robot otonom dalam skala besar di Skydio, kinerja dan pemeliharaan kode sangat penting untuk kasus penggunaan seperti komputer visi, estimasi keadaan, perencanaan gerak, dan kontrol. SymForce dibangun di atas manipulasi simbolik kemampuan perpustakaan SymPy. Matematika simbolik memungkinkan untuk pemahaman cepat, analisis interaktif, dan simbolik manipulasi seperti substitusi, pemecahan, dan diferensiasi. SymForce menambahkan geometri simbolis dan tipe kamera dengan Lie operasi grup, yang digunakan untuk membuat kelas waktu lari cepat secara otomatis dengan antarmuka yang identik. Dengan menggunakan satu simbol implementasi fungsi apa pun untuk menghasilkan kode runtime beberapa bahasa, Symforce meningkatkan siklus iterasi, meminimalkan kemungkinan bug, dan mencapai kinerja yang cocok atau
melebihi pendekatan state-of-the-art tanpa spesialisasi.

Singkatnya, Symforce adalah:

• Pustaka sumber terbuka dan gratis dengan:

– Implementasi simbolik geometri dan kamera
jenis dengan operasi grup Lie, dan runtime cepat
kelas dengan antarmuka yang identik,

– Pembuatan kode untuk mengubah fungsi simbolis arbitrer menjadi fungsi runtime yang terstruktur dan cepat,

– Pengoptimal ruang singgung cepat di C++ dan Python,

– Kode yang sangat berkinerja, modular, dan dapat diperluas,

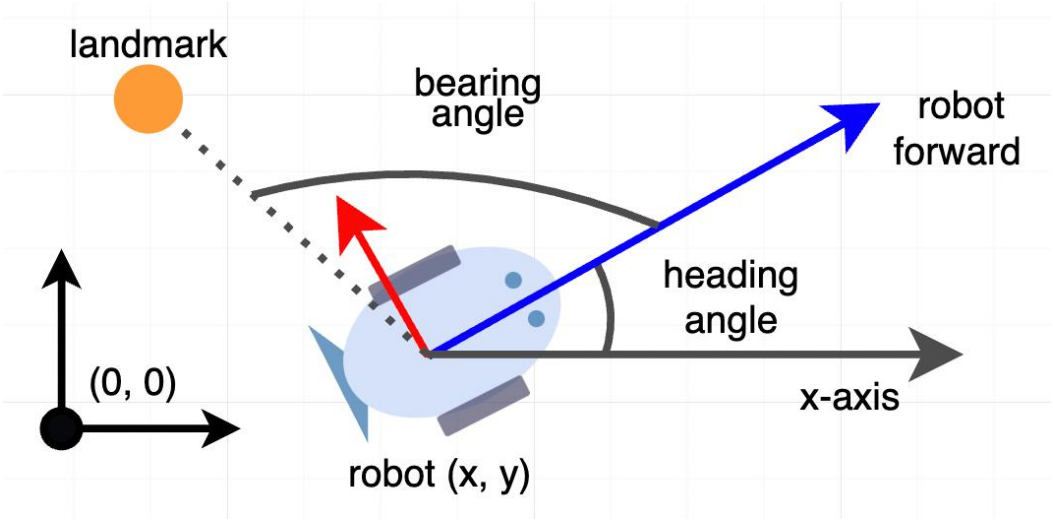
• Kontribusi baru untuk menghitung ruang singgung Jacobian secara otomatis, menghindari semua turunan tulisan tangan,

• Sebuah metode baru untuk menghindari singularitas dalam kompleks
ekspresi tanpa memperkenalkan percabangan,

• Sebuah eksposisi manfaat kecepatan yang diberikan oleh perataan
perhitungan lintas fungsi dan perkalian matriks,
terutama untuk mengungguli diferensiasi otomatis

3.0 ADDITIONAL SYSTEM REQUIREMENTS

3.1 System Description



Pada sistem robot yang digunakan adalah robot yang dapat memperkirakan pose pada beberapa langkah kedepan berdasarkan pegukuran kebisingan. Pengukuran yang harus dilakukan robot ini adalah sebagai berikut:

1. Jarak yang ditempuh

2. Sudut relatif menuju landmark

Pengukuran sudut relatif ditentukan berdasarkan arah maju robot.

4.0 GENERAL EQUIPMENT AND SOFTWARE

4.1 Equipment

Sistem ini menggunakan Symforce sebagai library untuk pengaplikasian metode robotik. Oleh karena itu agar library Symforce dapat berjalan, beberapa perlengkapan lain juga diperlukan untuk memastikan library symforce dapat bekerja dengan benar, perlengkapan-perlengkapan yang dibutuhkan dalam sistem ini adalah sebagai berikut:

1. Python versi 3.8, 3.9, 3.10, atau 3.11
2. C++ 14
3. Pypi v0.7.0
4. Apache 2.0

Setelah semua persyaratan diatas terpenuhi, maka library Symforce dapat diinstal dengan cara :

pip install symforce

Setelah itu dilanjutkan dengan menginstal numpy dengan cara :

pip install numpy

4.2 Software

Pada sistem ini software yang digunakan adalah sebuah IDE yang berfungsi untuk menuliskan baris kode perintah pada robot. Serta terminal yang berfungsi untuk menginstall semua perlengkapan yang dibutuhkan dan mengintall library symforce pada perangkat.

5.0 Code and Simulation

5.1 Import Library

```
import symforce

symforce.set_epsilon_to_symbol()

# ----------------------------------------------------------------------------
-
# Import Library
# ----------------------------------------------------------------------------
-
import numpy as np

from symforce import typing as T
from symforce.values import Values
import symforce.symbolic as sf
```

5.2 Value Initiation

```
def build_initial_values() -> T.Tuple[Values, int, int]:

    num_poses = 3
    num_landmarks = 3

    initial_values = Values(
        poses=[sf.Pose2.identity()] * num_poses,
        landmarks=[sf.V2(-2, 2), sf.V2(1, -3), sf.V2(5, 2)],
        distances=[1.7, 1.4],
        angles=np.deg2rad([[55, 245, -35], [95, 220, -20], [125, 220, -
20]]).tolist(),
        epsilon=sf.numeric_epsilon,
    )

    return initial_values, num_poses, num_landmarks
```

5.3 2D Residu

```
def bearing_residual(
    pose: sf.Pose2, landmark: sf.V2, angle: sf.Scalar, epsilon: sf.Scalar
) -> sf.V1:

    t_body = pose.inverse() * landmark
    predicted_angle = sf.atan2(t_body[1], t_body[0], epsilon=epsilon)
    return sf.V1(sf.wrap_angle(predicted_angle - angle))
```

5.4 2D Residu Distance

```
def odometry_residual(
    pose_a: sf.Pose2, pose_b: sf.Pose2, dist: sf.Scalar, epsilon: sf.Scalar
) -> sf.V1:

    return sf.V1((pose_b.t - pose_a.t).norm(epsilon=epsilon) - dist)


from symforce.opt.factor import Factor
```

5.5 Build Factor

```
def build_factors(num_poses: int, num_landmarks: int) -> T.Iterator[Factor]:

    for i in range(num_poses - 1):
        yield Factor(
            residual=odometry_residual,
            keys=[f"poses[{i}]", f"poses[{i + 1}]", f"distances[{i}]",
"epsilon"],
        )

    for i in range(num_poses):
        for j in range(num_landmarks):
            yield Factor(
                residual=bearing_residual,
                keys=[f"poses[{i}]", f"landmarks[{j}]", f"angles[{i}][{j}]",
"epsilon"],
            )
```

5.6 Main Function

```
def main() -> None:
    initial_values, num_poses, num_landmarks = build_initial_values()

    factors = build_factors(num_poses=num_poses,
num_landmarks=num_landmarks)

    optimized_keys = [f"poses[{i}]" for i in range(num_poses)]

    optimizer = Optimizer(
        factors=factors,
        optimized_keys=optimized_keys,
        debug_stats=True,  # Return problem stats for every iteration
        params=Optimizer.Params(verbose=True),  # Customize optimizer
behavior
    )

    result = optimizer.optimize(initial_values)
```

5.7 Print Function

```
   print(f"Num iterations: {len(result.iteration_stats) - 1}")
   print(f"Final error: {result.error():.6f}")

   for i, pose in enumerate(result.optimized_values["poses"]):
       print(f"Pose {i}: t = {pose.position()}, heading =
{pose.rotation().to_tangent()[0]}")

   from symforce.examples.robot_2d_localization.plotting import plot_solution

   plot_solution(optimizer, result)
```

5.8 Output Result

```
if __name__ == "__main__":
    main()
```

```
Num iterations: 8
Final error: 0.000220
Pose 0: t = [[-0.58303818]
 [-0.82449079]], heading = [1.073486]
Pose 1: t = [[ 1.01671023]
 [-0.23835618]], heading = [0.85760621]
Pose 2: t = [[1.79784992]
 [0.92055145]], heading = [0.67637098]
```