Code Generation from Unified Robot Description Format (URDF) for Accelerated Robotics

PAUL GESEL

Cppcon
The C++ Conference

2024

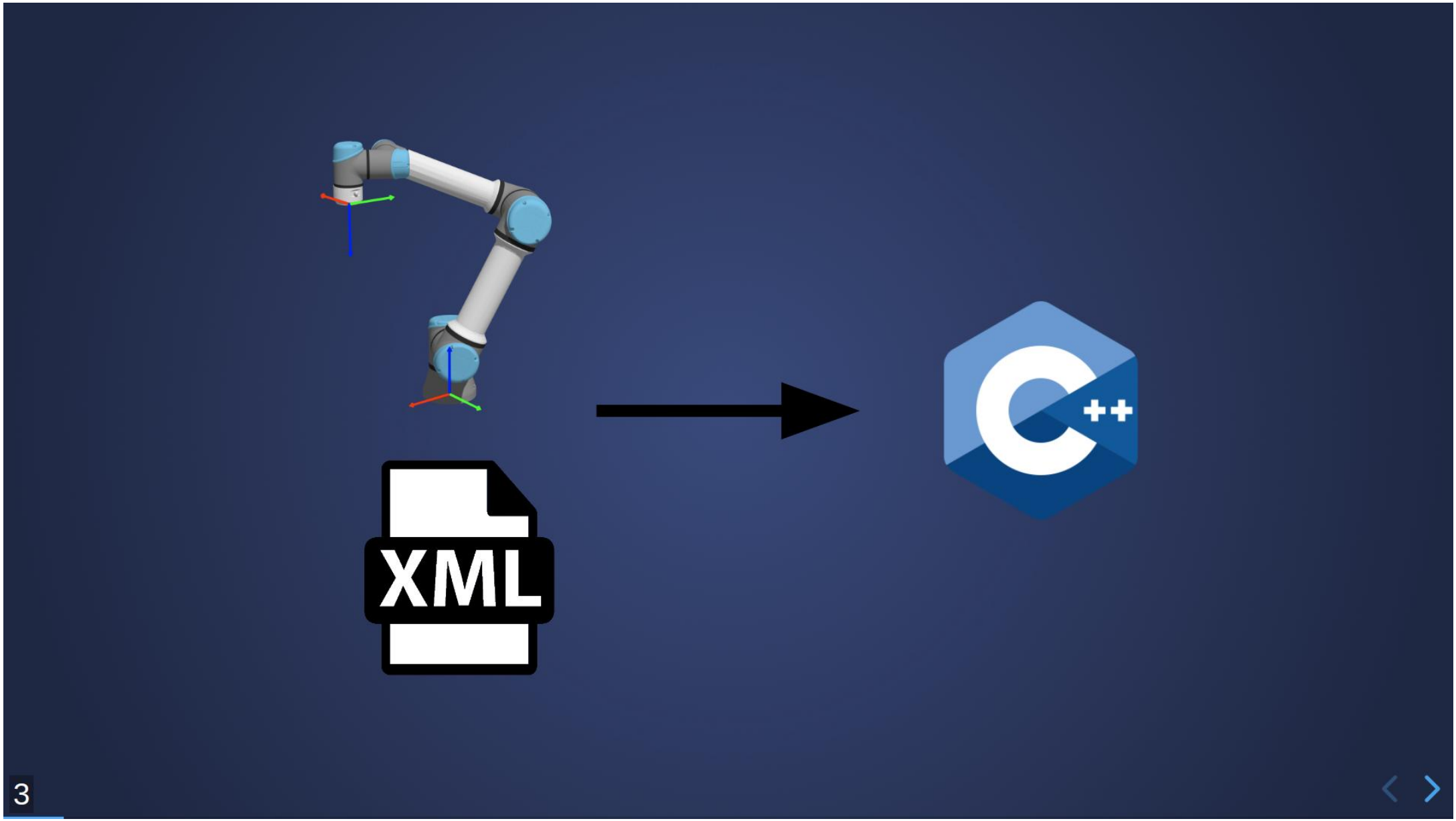
September 15 - 20

Introduction

1. I am a first time presenter and attender at CppCon

2. About me: I work as a robotics scientist at PickNik Robotics

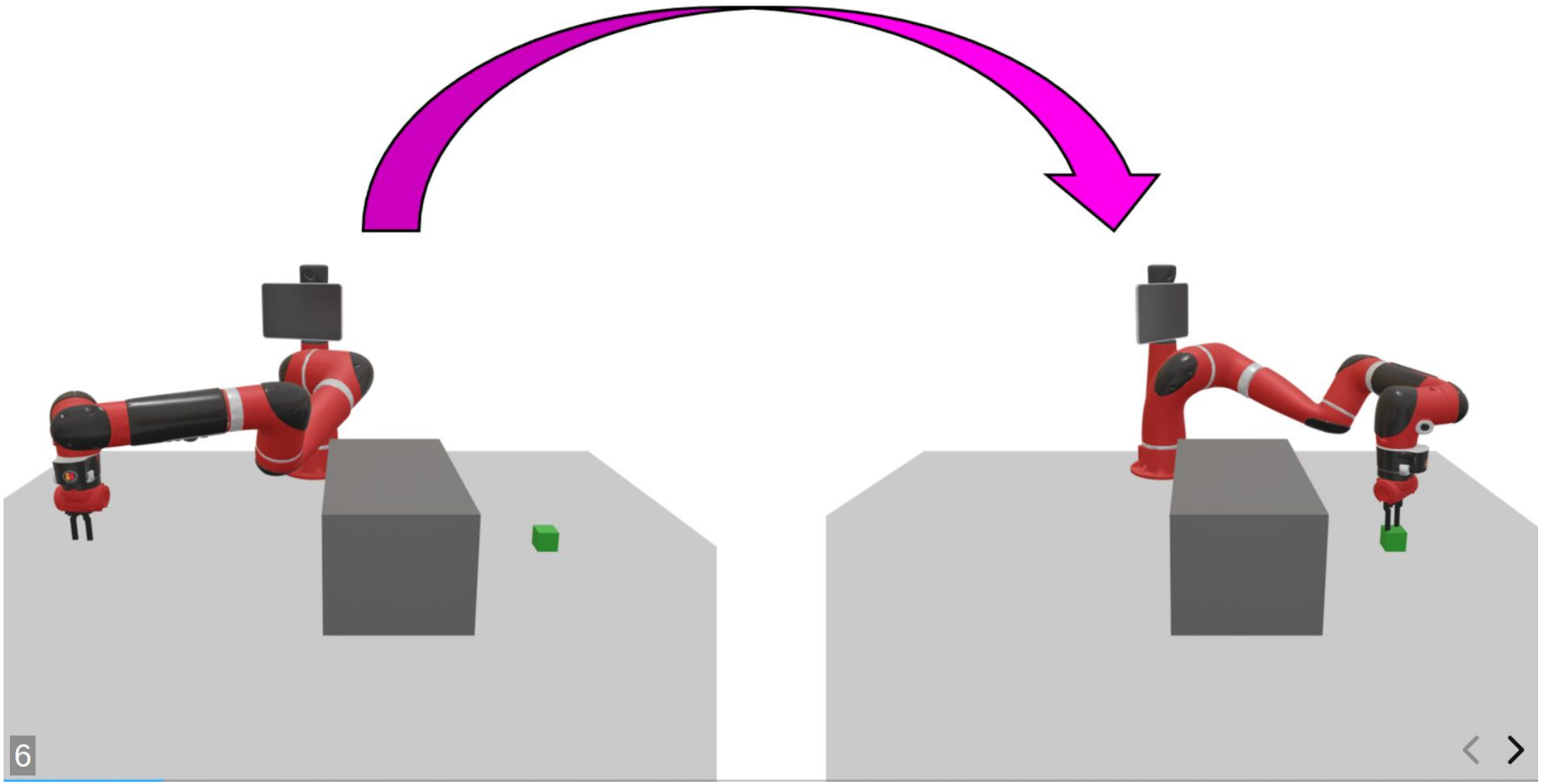
3. I earned my PhD in computer science from the University of New Hampshire on the topic of robot learning

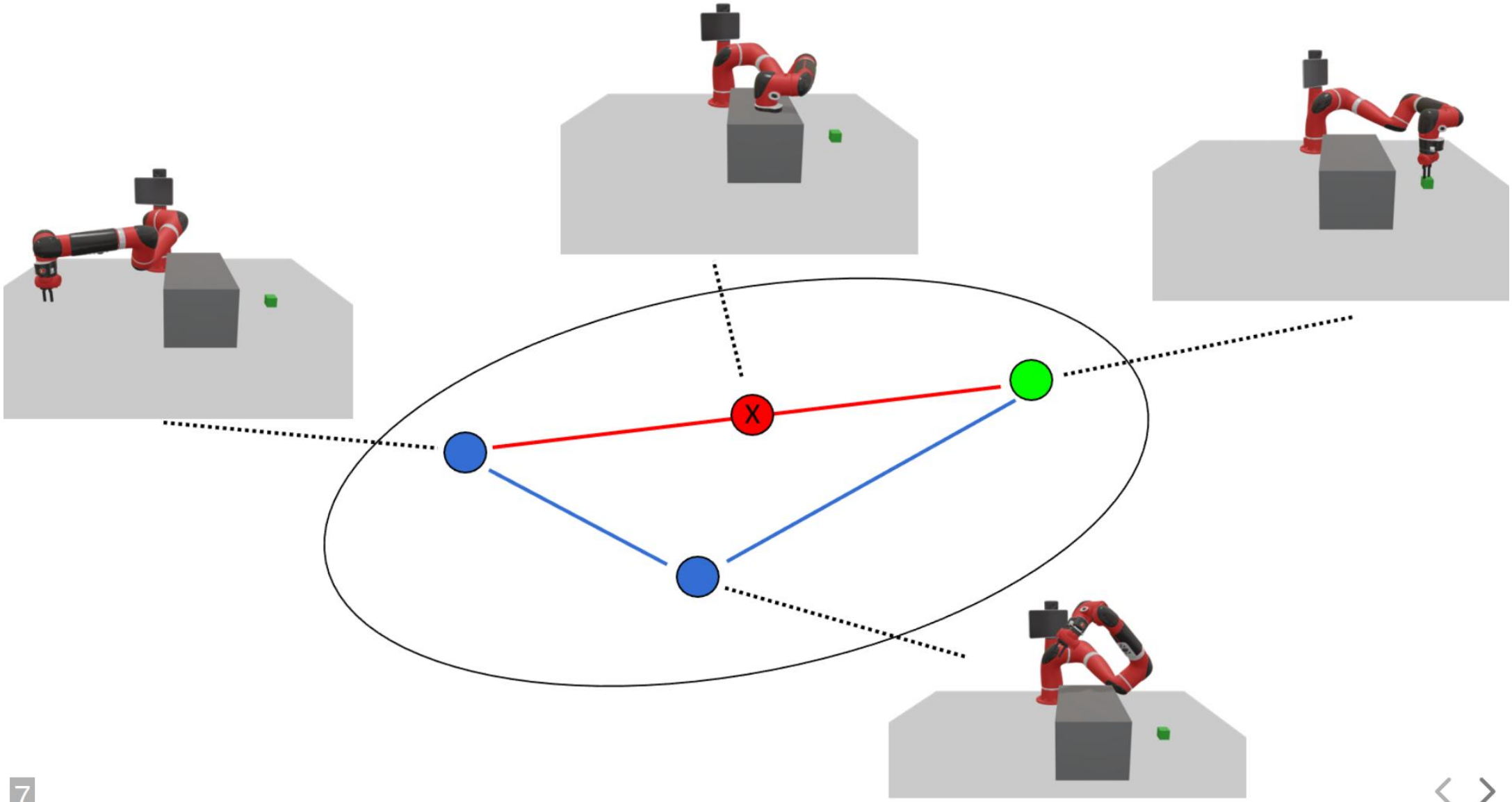

XML
C++
3

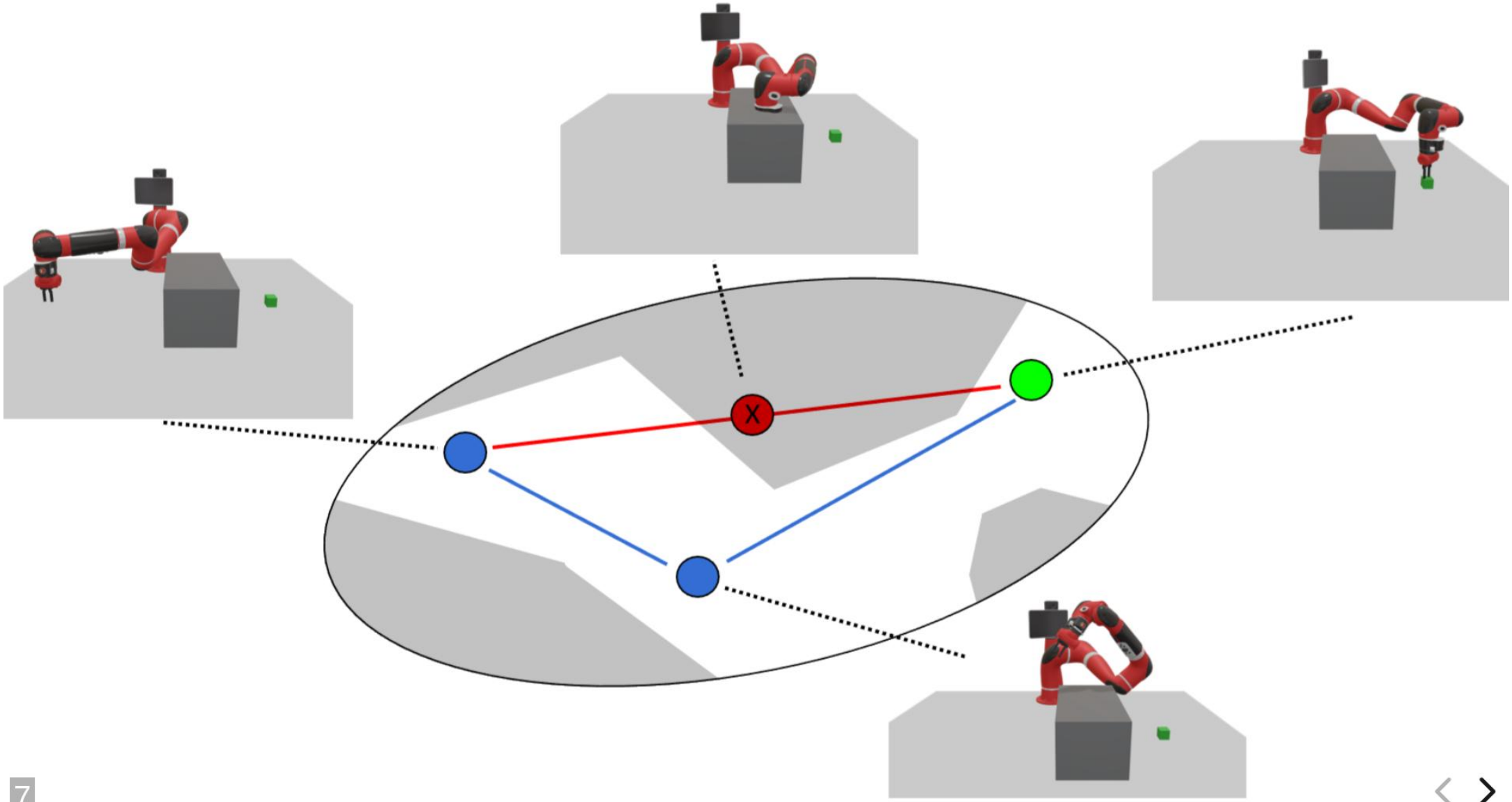
Outline

- Background
- Motivation
- Build Process
- Code Implementation
- Results
- Summary: lessons learned

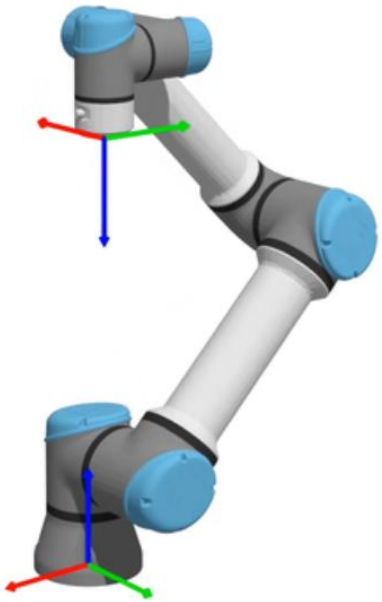
THE BACKGROUND


6

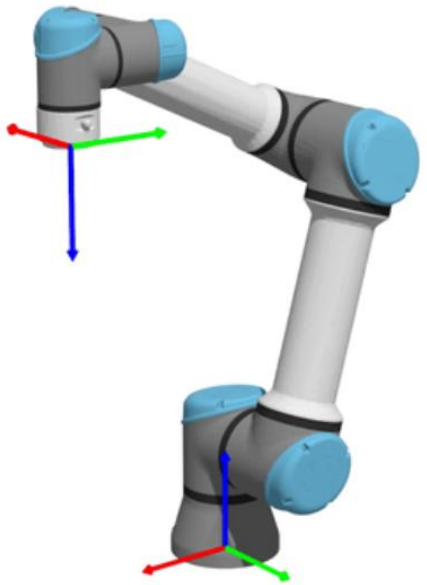

X


X

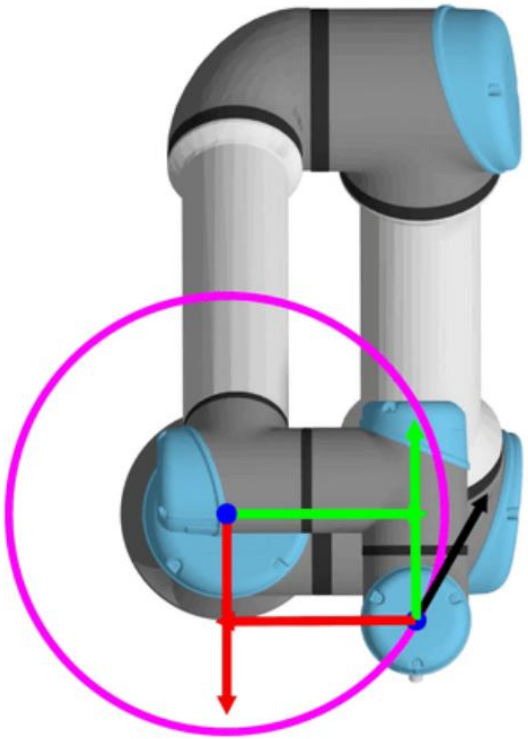
$$FK(q) = T_{l1}^{b} T_{l2}^{l1} T_{l3}^{l2} \dots$$



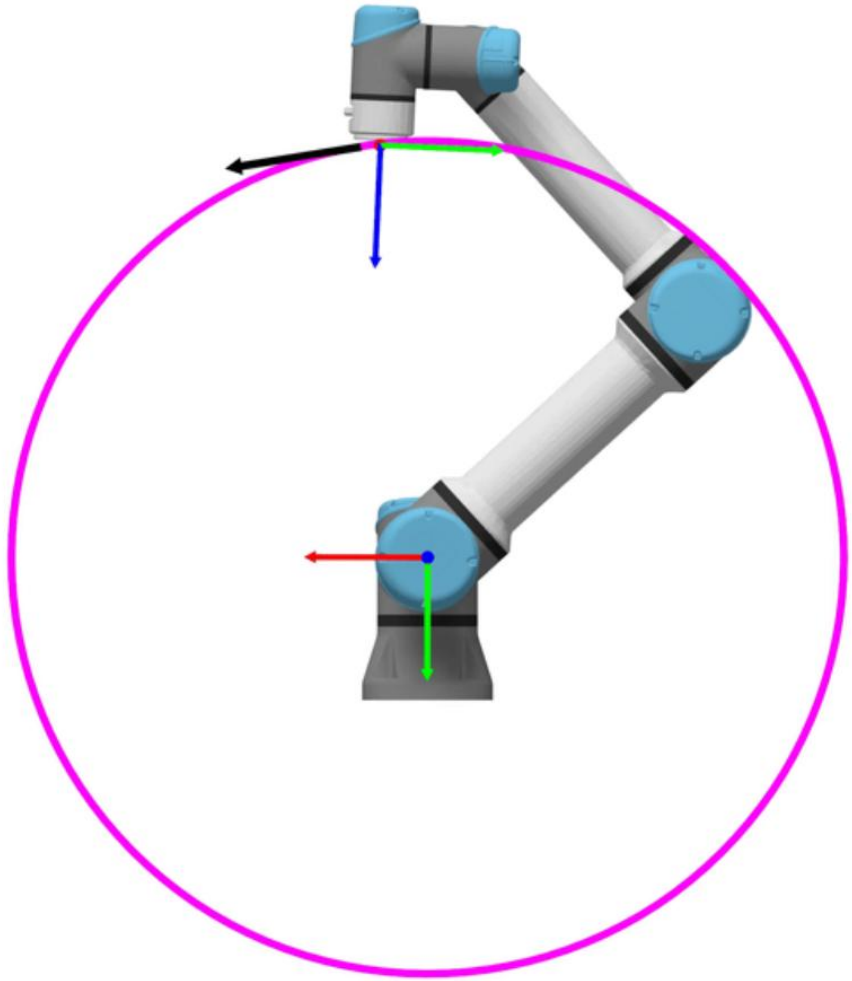
$$J(q) = \begin{bmatrix} \frac{\partial x}{\partial q_1} & \frac{\partial x}{\partial q_2} & \frac{\partial x}{\partial q_3} & \cdots \\ \frac{\partial y}{\partial q_1} & \frac{\partial y}{\partial q_2} & \frac{\partial y}{\partial q_3} & \cdots \\ \frac{\partial z}{\partial q_1} & \frac{\partial z}{\partial q_2} & \frac{\partial z}{\partial q_3} & \cdots \\ \vdots & \vdots & \vdots & \end{bmatrix}$$



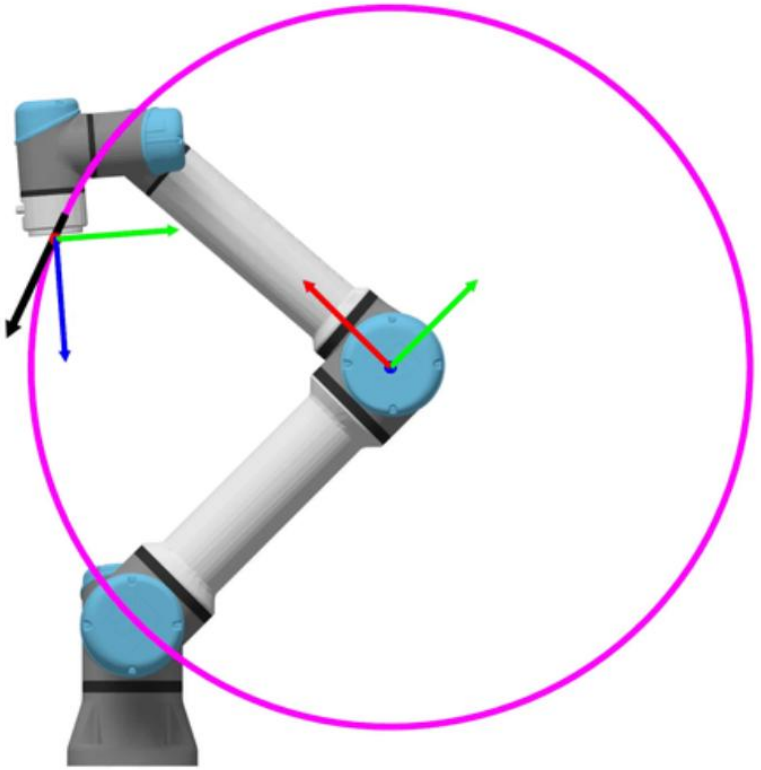
$$J(q) = \begin{bmatrix} \frac{\partial x}{\partial q_1} & \frac{\partial x}{\partial q_2} & \frac{\partial x}{\partial q_3} & \cdots \\ \frac{\partial y}{\partial q_1} & \frac{\partial y}{\partial q_2} & \frac{\partial y}{\partial q_3} & \cdots \\ \frac{\partial z}{\partial q_1} & \frac{\partial z}{\partial q_2} & \frac{\partial z}{\partial q_3} & \cdots \\ \vdots & \vdots & \vdots & \end{bmatrix}$$



$$J(q) = \begin{bmatrix} \frac{\partial x}{\partial q_1} & \frac{\partial x}{\partial q_2} & \frac{\partial x}{\partial q_3} & \cdots \\ \frac{\partial y}{\partial q_1} & \frac{\partial y}{\partial q_2} & \frac{\partial y}{\partial q_3} & \cdots \\ \frac{\partial z}{\partial q_1} & \frac{\partial z}{\partial q_2} & \frac{\partial z}{\partial q_3} & \cdots \\ \vdots & \vdots & \vdots & \end{bmatrix}$$

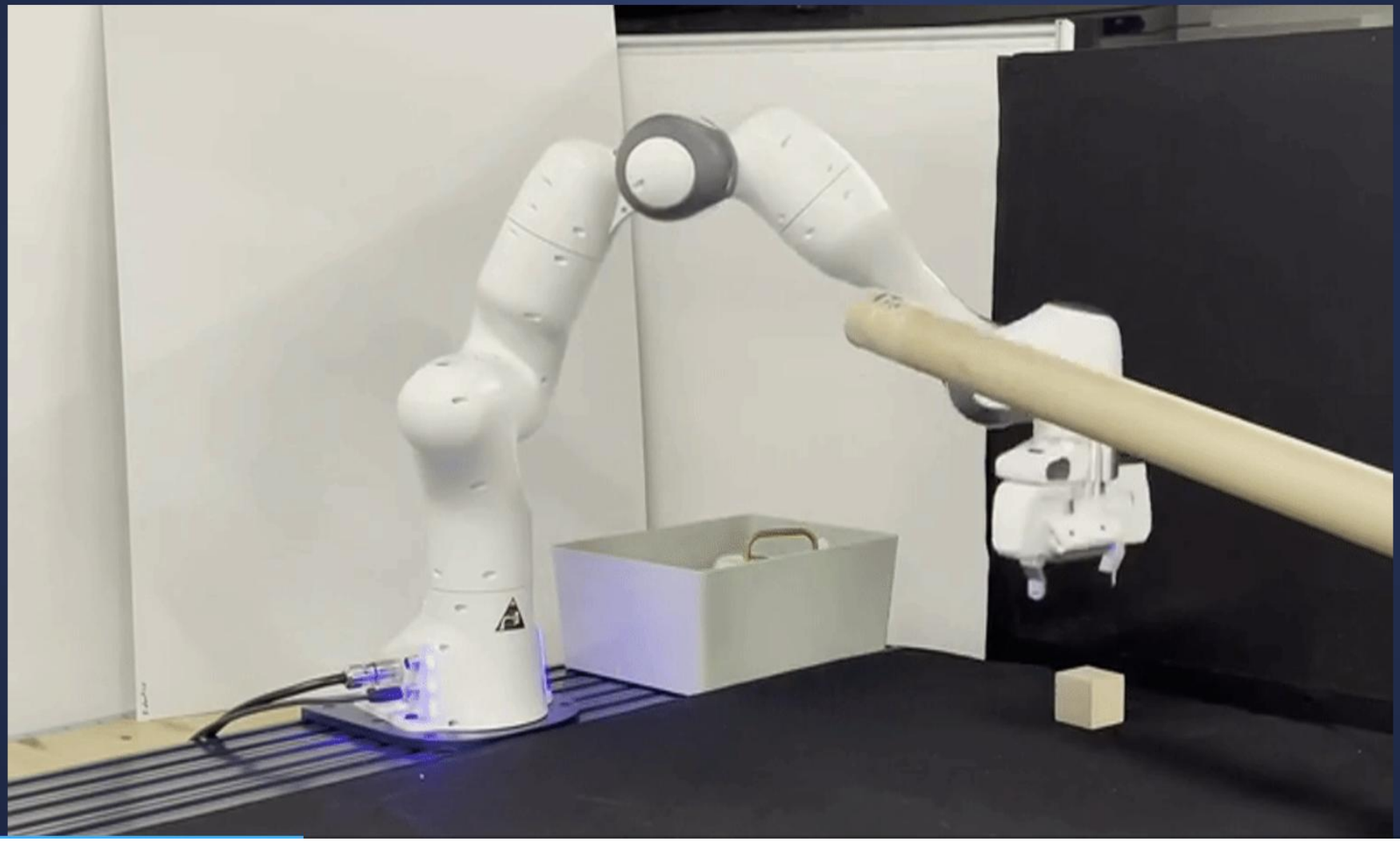


$$J(q) = \begin{bmatrix} \frac{\partial x}{\partial q_1} & \frac{\partial x}{\partial q_2} & \frac{\partial x}{\partial q_3} & \cdots \\ \frac{\partial y}{\partial q_1} & \frac{\partial y}{\partial q_2} & \frac{\partial y}{\partial q_3} & \cdots \\ \frac{\partial z}{\partial q_1} & \frac{\partial z}{\partial q_2} & \frac{\partial z}{\partial q_3} & \cdots \\ \vdots & \vdots & \vdots & \end{bmatrix}$$



MOTIVATION

Concept video



Motivation

- Some algorithms need to run at incredible speeds for optimal performance
- This is especially true for common subroutines used by expensive algorithms
- For example, robot path planning with RRT depends on nearest neighbor lookup and forward kinematics, and collision detection

Motivation

Motions in Microseconds via Vectorized Sampling-Based Planning

Wil Thomason[†], Zachary Kingston[†], and Lydia E. Kavraki

- Performance improvements of more than 500x over the state-of-the-art
- Compiler takes in standard Unified Robot Description Format (URDF) files and generates optimized code
- Setup data structure to optimize SIMD execution
- Skip unneeded computations like combining fixed joints, unrolling loops

Motivation

- Software written by hardware manufactures has an inherit advantage because it can be written with very specific platforms in mind and take advantage of hardware specific knowledge
- For example, robot design with spherical wrist can decompose inverse kinematics into two stages 1) find the wrist angles to achieve the desired orientation and 2) find the position value needed to find reach the desired Cartesian position
- Generic IK solvers cannot take advantage of this kind of knowledge in general
- One downside of hardware specific implementations is scalability
- Hardware generally does not change very often, other than attachments, like cameras and end effectors

Motivation

- Compiled code can be tested for memory allocations and real-time compatibility. Safety critical applications, such as surgical robots need to meet these requirements
- Allows for hardware specific CPU optimizations to be enabled since the code is source built on every machine
- Some advantages of code generation as opposed to a templated robot builder are 1) less likely to make a mistake building defining the robot 2) do not have to maintain code for each of your robots

THE BUILD PROCESS

```cmake
cmake_minimum_required(VERSION 3.22)
project(fast_forward_kinematics)

# create executable
add_executable(forward_kinematics_test forwardKinematicsTest.cpp)

# setup targets to generate C++ code
set(URDF_FILE "data/robot.urdf")
set(ROOT "base_link")
set(TIP "tool0")
generate_fast_forward_kinematics_library(${URDF_FILE} ${ROOT} ${TIP})

# link against generated code
target_link_libraries(forward_kinematics_test PUBLIC fast_forward_kinematics_library)
```

```cmake
cmake_minimum_required(VERSION 3.22)
project(fast_forward_kinematics)

# create executable
add_executable(forward_kinematics_test forwardKinematicsTest.cpp)

# setup targets to generate C++ code
set(URDF_FILE "data/robot.urdf")
set(ROOT "base_link")
set(TIP "tool0")
generate_fast_forward_kinematics_library(${URDF_FILE} ${ROOT} ${TIP})

# link against generated code
target_link_libraries(forward_kinematics_test PUBLIC fast_forward_kinematics_library)
```

```cmake
cmake_minimum_required(VERSION 3.22)
project(fast_forward_kinematics)

# create executable
add_executable(forward_kinematics_test forwardKinematicsTest.cpp)

# setup targets to generate C++ code
set(URDF_FILE "data/robot.urdf")
set(ROOT "base_link")
set(TIP "tool0")
generate_fast_forward_kinematics_library(${URDF_FILE} ${ROOT} ${TIP})

# link against generated code
target_link_libraries(forward_kinematics_test PUBLIC fast_forward_kinematics_library)
```

```cmake
cmake_minimum_required(VERSION 3.22)
project(fast_forward_kinematics)

# create executable
add_executable(forward_kinematics_test forwardKinematicsTest.cpp)

# setup targets to generate C++ code
set(URDF_FILE "data/robot.urdf")
set(ROOT "base_link")
set(TIP "tool0")
generate_fast_forward_kinematics_library(${URDF_FILE} ${ROOT} ${TIP})

# link against generated code
target_link_libraries(forward_kinematics_test PUBLIC fast_forward_kinematics_library)
```

```
function(generate_fast_forward_kinematics_library URDF_FILE ROOT_LINK TIP_LINK)

    find_package(Python REQUIRED COMPONENTS Interpreter)
    if (Python_Interpreter_FOUND)
        message(STATUS "Python executable: ${Python_EXECUTABLE}")
    else ()
        message(FATAL_ERROR "Python interpreter not found.")
    endif ()

    execute_process(
            COMMAND ${Python_EXECUTABLE} ${CMAKE_SOURCE_DIR}/scripts/get_num_joints.py
            ${URDF_FILE}
            ${ROOT_LINK}
            ${TIP_LINK}
            OUTPUT_VARIABLE FAST_FK_NUMBER_OF_JOINTS
            OUTPUT_STRIP_TRAILING_WHITESPACE
            COMMAND_ECHO STDOUT
    )

    include(ExternalProject)
    ExternalProject_Add(
            LBFGSpp
            PREFIX ${CMAKE_BINARY_DIR}/LBFGSpp
            GIT_REPOSITORY https://github.com/yixuan/LBFGSpp.git
            GIT_TAG v0.3.0
            CMAKE_ARGS -DCMAKE_INSTALL_PREFIX=${CMAKE_BINARY_DIR}
    )
    ExternalProject_Get_Property(LBFGSpp source_dir)
    set(LBFGSppIncludeDir ${source_dir}/include)

    add_custom_command(
            OUTPUT forward_kinematics_lib.cpp
            COMMAND ${Python_EXECUTABLE} ${CMAKE_SOURCE_DIR}/scripts/robot_gen.py
            ${URDF_FILE}
            ${CMAKE_SOURCE_DIR}/scripts/robot_config.cpp.template
            ${CMAKE_CURRENT_BINARY_DIR}/forward_kinematics_lib.cpp
```

```
function(generate_fast_forward_kinematics_library URDF_FILE ROOT_LINK TIP_LINK)

    find_package(Python REQUIRED COMPONENTS Interpreter)
    if (Python_Interpreter_FOUND)
        message(STATUS "Python executable: ${Python_EXECUTABLE}")
    else ()
        message(FATAL_ERROR "Python interpreter not found.")
    endif ()

    execute_process(
            COMMAND ${Python_EXECUTABLE} ${CMAKE_SOURCE_DIR}/scripts/get_num_joints.py
            ${URDF_FILE}
            ${ROOT_LINK}
            ${TIP_LINK}
            OUTPUT_VARIABLE FAST_FK_NUMBER_OF_JOINTS
            OUTPUT_STRIP_TRAILING_WHITESPACE
            COMMAND_ECHO STDOUT
    )

    include(ExternalProject)
    ExternalProject_Add(
            LBFGSpp
            PREFIX ${CMAKE_BINARY_DIR}/LBFGSpp
            GIT_REPOSITORY https://github.com/yixuan/LBFGSpp.git
            GIT_TAG v0.3.0
            CMAKE_ARGS -DCMAKE_INSTALL_PREFIX=${CMAKE_BINARY_DIR}
    )
    ExternalProject_Get_Property(LBFGSpp source_dir)
    set(LBFGSppIncludeDir ${source_dir}/include)

    add_custom_command(
            OUTPUT forward_kinematics_lib.cpp
            COMMAND ${Python_EXECUTABLE} ${CMAKE_SOURCE_DIR}/scripts/robot_gen.py
            ${URDF_FILE}
            ${CMAKE_SOURCE_DIR}/scripts/robot_config.cpp.template
            ${CMAKE_CURRENT_BINARY_DIR}/forward_kinematics_lib.cpp
```

```python
from urdf_parser_py import urdf
import argparse


def run():
    parser = argparse.ArgumentParser()
    parser.add_argument('urdf_file')
    parser.add_argument('root_link_name')
    parser.add_argument('tip_link_name')
    args = parser.parse_args()

    root_link_name = args.root_link_name
    tip_link_name = args.tip_link_name
    with open(args.urdf_file) as f:
        robot = urdf.Robot.from_xml_string(f.read())

    joint_names = []
    while tip_link_name != root_link_name:
        tip_joint_name, tip_link_name = robot.parent_map[tip_link_name]
        joint_names.append(tip_joint_name)

    print(f"FAST_FK_NUMBER_OF_JOINTS={len(joint_names)}", end="")


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

```xml
<?xml version="1.0" ?>
<robot name="ur5e">
  <link name="world"/>
  <link name="base_link"/>
  <link name="base_link_inertia">
    <visual>
      <origin rpy="0 0 3.141592653589793" xyz="0 0 0"/>
      <geometry>
        <mesh filename="package://ur_description/meshes/ur5e/visual/base.dae"/>
      </geometry>
    </visual>
    <collision>
      <origin rpy="0 0 3.141592653589793" xyz="0 0 0"/>
      <geometry>
        <mesh filename="package://ur_description/meshes/ur5e/collision/base.stl"/>
      </geometry>
    </collision>
    <inertial>
      <mass value="4.0"/>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <inertia ixx="0.00443333156" ixy="0.0" ixz="0.0" iyy="0.00443333156" iyz="0.0" izz="0.0072"/>
    </inertial>
  </link>
  <link name="shoulder_link">
    <visual>
      <origin rpy="0 0 3.141592653589793" xyz="0 0 0"/>
      <geometry>
        <mesh filename="package://ur_description/meshes/ur5e/visual/shoulder.dae"/>
      </geometry>
    </visual>
    <collision>
      <origin rpy="0 0 3.141592653589793" xyz="0 0 0"/>
      <geometry>
        <mesh filename="package://ur_description/meshes/ur5e/collision/shoulder.stl"/>
      </geometry>
    </collision>
```

```
function(generate_fast_forward_kinematics_library URDF_FILE ROOT_LINK TIP_LINK)

    find_package(Python REQUIRED COMPONENTS Interpreter)
    if (Python_Interpreter_FOUND)
        message(STATUS "Python executable: ${Python_EXECUTABLE}")
    else ()
        message(FATAL_ERROR "Python interpreter not found.")
    endif ()

    execute_process(
            COMMAND ${Python_EXECUTABLE} ${CMAKE_SOURCE_DIR}/scripts/get_num_joints.py
            ${URDF_FILE}
            ${ROOT_LINK}
            ${TIP_LINK}
            OUTPUT_VARIABLE FAST_FK_NUMBER_OF_JOINTS
            OUTPUT_STRIP_TRAILING_WHITESPACE
            COMMAND_ECHO STDOUT
    )

    include(ExternalProject)
    ExternalProject_Add(
            LBFGSpp
            PREFIX ${CMAKE_BINARY_DIR}/LBFGSpp
            GIT_REPOSITORY https://github.com/yixuan/LBFGSpp.git
            GIT_TAG v0.3.0
            CMAKE_ARGS -DCMAKE_INSTALL_PREFIX=${CMAKE_BINARY_DIR}
    )
    ExternalProject_Get_Property(LBFGSpp source_dir)
    set(LBFGSppIncludeDir ${source_dir}/include)

    add_custom_command(
            OUTPUT forward_kinematics_lib.cpp
            COMMAND ${Python_EXECUTABLE} ${CMAKE_SOURCE_DIR}/scripts/robot_gen.py
            ${URDF_FILE}
            ${CMAKE_SOURCE_DIR}/scripts/robot_config.cpp.template
            ${CMAKE_CURRENT_BINARY_DIR}/forward_kinematics_lib.cpp
```

```
7         message(FATAL_ERROR "Python interpreter not found.")
8     endif ()
9
10    execute_process(
11            COMMAND ${Python_EXECUTABLE} ${CMAKE_SOURCE_DIR}/scripts/get_num_joints.py
12            ${URDF_FILE}
13            ${ROOT_LINK}
14            ${TIP_LINK}
15            OUTPUT_VARIABLE FAST_FK_NUMBER_OF_JOINTS
16            OUTPUT_STRIP_TRAILING_WHITESPACE
17            COMMAND_ECHO STDOUT
18    )
19
20    include(ExternalProject)
21    ExternalProject_Add(
22            LBFGSpp
23            PREFIX ${CMAKE_BINARY_DIR}/LBFGSpp
24            GIT_REPOSITORY https://github.com/yixuan/LBFGSpp.git
25            GIT_TAG v0.3.0
26            CMAKE_ARGS -DCMAKE_INSTALL_PREFIX=${CMAKE_BINARY_DIR}
27    )
28    ExternalProject_Get_Property(LBFGSpp source_dir)
29    set(LBFGSppIncludeDir ${source_dir}/include)
30
31    add_custom_command(
32            OUTPUT forward_kinematics_lib.cpp
33            COMMAND ${Python_EXECUTABLE} ${CMAKE_SOURCE_DIR}/scripts/robot_gen.py
34            ${URDF_FILE}
35            ${CMAKE_SOURCE_DIR}/scripts/robot_config.cpp.template
36            ${CMAKE_CURRENT_BINARY_DIR}/forward_kinematics_lib.cpp
37            ${ROOT_LINK}
38            ${TIP_LINK}
39            DEPENDS ${URDF_FILE} ${CMAKE_SOURCE_DIR}/scripts/robot_config.cpp.template
40            COMMENT
41            "Running `${PYTHON_EXECUTABLE} ${CMAKE_SOURCE_DIR}/scripts/robot_gen.py
42                    ${URDF_FILE}
```

```
        LBFGSpp
        PREFIX ${CMAKE_BINARY_DIR}/LBFGSpp
        GIT_REPOSITORY https://github.com/yixuan/LBFGSpp.git
        GIT_TAG v0.3.0
        CMAKE_ARGS -DCMAKE_INSTALL_PREFIX=${CMAKE_BINARY_DIR}
)
ExternalProject_Get_Property(LBFGSpp source_dir)
set(LBFGSppIncludeDir ${source_dir}/include)

add_custom_command(
        OUTPUT forward_kinematics_lib.cpp
        COMMAND ${Python_EXECUTABLE} ${CMAKE_SOURCE_DIR}/scripts/robot_gen.py
        ${URDF_FILE}
        ${CMAKE_SOURCE_DIR}/scripts/robot_config.cpp.template
        ${CMAKE_CURRENT_BINARY_DIR}/forward_kinematics_lib.cpp
        ${ROOT_LINK}
        ${TIP_LINK}
        DEPENDS ${URDF_FILE} ${CMAKE_SOURCE_DIR}/scripts/robot_config.cpp.template
        COMMENT
        "Running `${PYTHON_EXECUTABLE} ${CMAKE_SOURCE_DIR}/scripts/robot_gen.py
                ${URDF_FILE}
                ${CMAKE_SOURCE_DIR}/scripts/robot_config.cpp.template
                ${CMAKE_CURRENT_BINARY_DIR}/forward_kinematics_test.cpp
                ${ROOT_LINK}
                ${TIP_LINK}`"
        VERBATIM
)
add_custom_target(code_generation DEPENDS forward_kinematics_lib.cpp)

add_library(fast_forward_kinematics_library SHARED forward_kinematics_lib.cpp)
add_dependencies(fast_forward_kinematics_library code_generation)
add_dependencies(fast_forward_kinematics_library LBFGSpp)

find_package(Eigen3 3.3 NO_MODULE)
target_compile_definitions(fast_forward_kinematics_library PUBLIC "${FAST_FK_NUMBER_OF_JOINTS}")
target_include_directories(fast_forward_kinematics_library PUBLIC ${CMAKE_SOURCE_DIR}/include)
target_include_directories(fast_forward_kinematics_library PUBLIC ${LBFGSppIncludeDir})
```

```python
from jinja2 import Template
from urdf_parser_py import urdf

import argparse
import numpy as np


def run():
    parser = argparse.ArgumentParser()
    parser.add_argument('urdf_file')
    parser.add_argument('fk_template')
    parser.add_argument('fk_output_file')
    parser.add_argument('root_link_name')
    parser.add_argument('tip_link_name')
    args = parser.parse_args()

    root_link_name = args.root_link_name
    tip_link_name = args.tip_link_name
    with open(args.urdf_file) as f:
        robot = urdf.Robot.from_xml_string(f.read())

    def get_transform(joint):
        rpy = joint.origin.rpy
        xyz = joint.origin.xyz
        T = np.eye(4)

        yaw = np.array([[np.cos(rpy[2]), np.sin(rpy[2]), 0],
                        [-np.sin(rpy[2]), np.cos(rpy[2]), 0],
                        [0, 0, 1]])

        pitch = np.array([[np.cos(rpy[1]), 0, np.sin(rpy[1])],
                          [0, 1, 0],
                          [-np.sin(rpy[1]), 0, np.cos(rpy[1])]])

        roll = np.array([[1, 0, 0],
                         [0, np.cos(rpy[0]), np.sin(rpy[0])],
```

```python
from jinja2 import Template
from urdf_parser_py import urdf

import argparse
import numpy as np


def run():
    parser = argparse.ArgumentParser()
    parser.add_argument('urdf_file')
    parser.add_argument('fk_template')
    parser.add_argument('fk_output_file')
    parser.add_argument('root_link_name')
    parser.add_argument('tip_link_name')
    args = parser.parse_args()

    root_link_name = args.root_link_name
    tip_link_name = args.tip_link_name
    with open(args.urdf_file) as f:
        robot = urdf.Robot.from_xml_string(f.read())

    def get_transform(joint):
        rpy = joint.origin.rpy
        xyz = joint.origin.xyz
        T = np.eye(4)

        yaw = np.array([[np.cos(rpy[2]), np.sin(rpy[2]), 0],
                        [-np.sin(rpy[2]), np.cos(rpy[2]), 0],
                        [0, 0, 1]])

        pitch = np.array([[np.cos(rpy[1]), 0, np.sin(rpy[1])],
                          [0, 1, 0],
                          [-np.sin(rpy[1]), 0, np.cos(rpy[1])]])

        roll = np.array([[1, 0, 0],
                         [0, np.cos(rpy[0]), np.sin(rpy[0])],
```

```
14      parser.add_argument('tip_link_name')
15      args = parser.parse_args()
16
17      root_link_name = args.root_link_name
18      tip_link_name = args.tip_link_name
19      with open(args.urdf_file) as f:
20          robot = urdf.Robot.from_xml_string(f.read())
21
22      def get_transform(joint):
23          rpy = joint.origin.rpy
24          xyz = joint.origin.xyz
25          T = np.eye(4)
26
27          yaw = np.array([[np.cos(rpy[2]), np.sin(rpy[2]), 0],
28                          [-np.sin(rpy[2]), np.cos(rpy[2]), 0],
29                          [0, 0, 1]])
30
31          pitch = np.array([[np.cos(rpy[1]), 0, np.sin(rpy[1])],
32                            [0, 1, 0],
33                            [-np.sin(rpy[1]), 0, np.cos(rpy[1])]])
34
35          roll = np.array([[1, 0, 0],
36                           [0, np.cos(rpy[0]), np.sin(rpy[0])],
37                           [0, -np.sin(rpy[0]), np.cos(rpy[0])]])
38          T[:3, :3] = yaw @ pitch @ roll
39          T[:3, 3] = xyz
40
41          return T
42
43      rotations = []
44      offsets = []
45      types = []
46      joint_names = []
47      while tip_link_name != root_link_name:
48          tip_joint_name, tip_link_name = robot.parent_map[tip_link_name]
49          joint_names.append(tip_joint_name)
```

```
                        [-np.sin(rpy[2]), np.cos(rpy[2]), 0],
                        [0, 0, 1]])

        pitch = np.array([[np.cos(rpy[1]), 0, np.sin(rpy[1])],
                          [0, 1, 0],
                          [-np.sin(rpy[1]), 0, np.cos(rpy[1])]])

        roll = np.array([[1, 0, 0],
                         [0, np.cos(rpy[0]), np.sin(rpy[0])],
                         [0, -np.sin(rpy[0]), np.cos(rpy[0])]])
        T[:3, :3] = yaw @ pitch @ roll
        T[:3, 3] = xyz

        return T

    rotations = []
    offsets = []
    types = []
    joint_names = []
    while tip_link_name != root_link_name:
        tip_joint_name, tip_link_name = robot.parent_map[tip_link_name]
        joint_names.append(tip_joint_name)

    joint_names.reverse()

    T_fixed = np.eye(4)
    for joint_name in joint_names:
        joint = robot.joint_map[joint_name]
        if joint.type == 'fixed':
            T_fixed = T_fixed * get_transform(joint)
        elif joint.type == 'revolute':
            # TODO: need to add joint limits?
            T = get_transform(joint) @ T_fixed
            T_fixed = np.eye(4)
            rotations.append(T[:3, :3])
            offsets.append(T[:3, 3])
            types.append('revolute')
```

```
        tip_joint_name, tip_link_name = robot.parent_map[tip_link_name]
        joint_names.append(tip_joint_name)

    joint_names.reverse()

    T_fixed = np.eye(4)
    for joint_name in joint_names:
        joint = robot.joint_map[joint_name]
        if joint.type == 'fixed':
            T_fixed = T_fixed * get_transform(joint)
        elif joint.type == 'revolute':
            # TODO: need to add joint limits?
            T = get_transform(joint) @ T_fixed
            T_fixed = np.eye(4)
            rotations.append(T[:3, :3])
            offsets.append(T[:3, 3])
            types.append('revolute')
        elif joint.type == 'continuous':
            T = get_transform(joint) @ T_fixed
            T_fixed = np.eye(4)
            rotations.append(T[:3, :3])
            offsets.append(T[:3, 3])
            types.append('revolute')
        elif joint.type == 'prismatic':
            # TODO: need to add joint limits?
            T = get_transform(joint) @ T_fixed
            T_fixed = np.eye(4)
            rotations.append(T[:3, :3])
            offsets.append(T[:3, 3])
            types.append('prismatic')
        else:
            raise Exception(f"joint type {joint.type} in URDF not supported")

    # truncate near zero values
    offsets = [val * (np.abs(val) > 1E-5) for val in offsets]
    rotations = [val * (np.abs(val) > 1E-5) for val in rotations]
```

```
            # TODO: need to add joint limits?
            T = get_transform(joint) @ T_fixed
            T_fixed = np.eye(4)
            rotations.append(T[:3, :3])
            offsets.append(T[:3, 3])
            types.append('revolute')
        elif joint.type == 'continuous':
            T = get_transform(joint) @ T_fixed
            T_fixed = np.eye(4)
            rotations.append(T[:3, :3])
            offsets.append(T[:3, 3])
            types.append('revolute')
        elif joint.type == 'prismatic':
            # TODO: need to add joint limits?
            T = get_transform(joint) @ T_fixed
            T_fixed = np.eye(4)
            rotations.append(T[:3, :3])
            offsets.append(T[:3, 3])
            types.append('prismatic')
        else:
            raise Exception(f"joint type {joint.type} in URDF not supported")

    # truncate near zero values
    offsets = [val * (np.abs(val) > 1E-5) for val in offsets]
    rotations = [val * (np.abs(val) > 1E-5) for val in rotations]

    with open(args.fk_template, 'r') as f:
        j2_template = Template(f.read())
        code = j2_template.render({'rotations': rotations, 'offsets': offsets, 'types': types}, trim_blocks=True)

    with open(args.fk_output_file, 'w') as f:
        f.write(code)


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

```
#include "fast_kinematics.hpp"

namespace fast_fk::internal {
    // sin(t) cos(t)  px py pz R11, R12, R13...
    constexpr int data_size = 2 + 3 + 9+2; // 16

  void forward_kinematics_internal(float *input_data, std::size_t /*size*/) {

      // row major
      constexpr std::array<float, {{rotations|length}}* 9> R_all = {
              {%- for rotation in rotations %}
              {%- for row in rotation %}
              {%- for val in row %} {{val}},
              {%- endfor %}
              {% endfor %}
              {%- endfor -%}
      };

      constexpr std::array<float, {{rotations|length}}* 3> offset_all = {
              {%- for offset in offsets %}
              {%- for val in offset %} {{val}},
              {%- endfor %}
              {% endfor -%}
      };

      float tmp1;
      float tmp2;
      float tmp3;

      {% for type in types %}
      {
          constexpr std::size_t ind = {{loop.index0}};

          // R_fixed is the rotation from joint_i+1 in frame joint_i (row major)
          const float &R11_fixed = R_all[ind * 9 + 0];
          const float &R21_fixed = R_all[ind * 9 + 1];
```

```cpp
#include "fast_kinematics.hpp"

namespace fast_fk::internal {
    // sin(t) cos(t)  px py pz R11, R12, R13...
    constexpr int data_size = 2 + 3 + 9 + 2; // 16

    void forward_kinematics_internal(float *input_data, std::size_t /*size*/) {

        // row major
        constexpr std::array<float, 6 * 9> R_all = { -1.0, -0.0, 0.0,
                                                     0.0, -1.0, 0.0,
                                                     0.0, 0.0, 1.0,
                                                     0.9999999999920636, -0.0, 0.0,
                                                     -0.0, -0.0007013484622411433, 0.9999997540472005,
                                                     0.0, -0.999999754055137, -0.0007013484622467095,
                                                     0.9999993163649181, 0.0, -0.001169289425224364,
                                                     -0.0, 0.9999994508798468, 0.0010479601216713834,
                                                     0.001169294705420077, -0.00104795423012248, 0.9999987672701519,
                                                     0.9999999072555382, -1.0253697707014652e-05, 0.0004305621635032162
                                                     0.0, 0.9999720394582239, 0.007477984507041285,
                                                     -0.0004306268017444494, -0.007477980785065483, 0.9999719467884766,
                                                     0.9999999999999949, -0.0, -0.0,
                                                     0.0, 0.001812660647159262, 0.9999983571293345,
                                                     0.0, -0.9999983571293396, 0.001812660647159271,
                                                     0.9999999999999998, 0.0, -0.0,
                                                     -0.0, 0.0011436177325240319, -0.9999993460690269,
                                                     -0.0, 0.9999993460690271, 0.001143617732524032,
        };

        constexpr std::array<float, 6 * 3> offset_all = { 0.0, 0.0, 0.1625702965797758,
                                                          0.000182214465989093, 0.0, 0.0,
                                                          -0.4249817627044961, 0.0, 0.0,
                                                          -0.3921666446509172, -0.0009975307642066673, 0.133391995638352
                                                          0.0, -0.09959821611958637, 0.0001805380634879481,
                                                          7.603964784130673e-05, 0.09950302422228456, 0.000113793497353
        };
```

```
#include <cstdio>

namespace fast_fk::internal {
    // sin(t) cos(t)  px py pz R11, R12, R13...
    constexpr int data_size = 2 + 3 + 9 + 2; // 16
    constexpr int num_of_joints = {{rotations|length}}; // 16

    __global__ void forward_kinematics_internal_kernel(float *input_data, std::size_t total_size) {
        unsigned int idx = blockIdx.x * blockDim.x + threadIdx.x;
        if (idx * data_size * num_of_joints >= total_size) {
            return;
        }
        input_data += (idx * data_size*num_of_joints);

        // row major
        constexpr float R_all[{{rotations|length}}* 9] = {
                {%- for rotation in rotations %}
                {%- for row in rotation %}
                {%- for val in row %} {{val}},
                {%- endfor %}
                {% endfor %}
                {%- endfor -%}
        };

        constexpr float offset_all[{{rotations|length}}* 3] = {
                {%- for offset in offsets %}
                {%- for val in offset %} {{val}},
                {%- endfor %}
                {% endfor -%}
        };

        float tmp1;
        float tmp2;
        float tmp3;

        {% for type in types %}
```

```
        LBFGSpp
        PREFIX ${CMAKE_BINARY_DIR}/LBFGSpp
        GIT_REPOSITORY https://github.com/yixuan/LBFGSpp.git
        GIT_TAG v0.3.0
        CMAKE_ARGS -DCMAKE_INSTALL_PREFIX=${CMAKE_BINARY_DIR}
)
ExternalProject_Get_Property(LBFGSpp source_dir)
set(LBFGSppIncludeDir ${source_dir}/include)

add_custom_command(
        OUTPUT forward_kinematics_lib.cpp
        COMMAND ${Python_EXECUTABLE} ${CMAKE_SOURCE_DIR}/scripts/robot_gen.py
        ${URDF_FILE}
        ${CMAKE_SOURCE_DIR}/scripts/robot_config.cpp.template
        ${CMAKE_CURRENT_BINARY_DIR}/forward_kinematics_lib.cpp
        ${ROOT_LINK}
        ${TIP_LINK}
        DEPENDS ${URDF_FILE} ${CMAKE_SOURCE_DIR}/scripts/robot_config.cpp.template
        COMMENT
        "Running `${PYTHON_EXECUTABLE} ${CMAKE_SOURCE_DIR}/scripts/robot_gen.py
                ${URDF_FILE}
                ${CMAKE_SOURCE_DIR}/scripts/robot_config.cpp.template
                ${CMAKE_CURRENT_BINARY_DIR}/forward_kinematics_test.cpp
                ${ROOT_LINK}
                ${TIP_LINK}`"
        VERBATIM
)
add_custom_target(code_generation DEPENDS forward_kinematics_lib.cpp)

add_library(fast_forward_kinematics_library SHARED forward_kinematics_lib.cpp)
add_dependencies(fast_forward_kinematics_library code_generation)
add_dependencies(fast_forward_kinematics_library LBFGSpp)

find_package(Eigen3 3.3 NO_MODULE)
target_compile_definitions(fast_forward_kinematics_library PUBLIC "${FAST_FK_NUMBER_OF_JOINTS}")
target_include_directories(fast_forward_kinematics_library PUBLIC ${CMAKE_SOURCE_DIR}/include)
target_include_directories(fast_forward_kinematics_library PUBLIC ${LBFGSppIncludeDir})
```

```
            ${URDF_FILE}
            ${CMAKE_SOURCE_DIR}/scripts/robot_config.cpp.template
            ${CMAKE_CURRENT_BINARY_DIR}/forward_kinematics_lib.cpp
            ${ROOT_LINK}
            ${TIP_LINK}
            DEPENDS ${URDF_FILE} ${CMAKE_SOURCE_DIR}/scripts/robot_config.cpp.template
            COMMENT
            "Running `${PYTHON_EXECUTABLE} ${CMAKE_SOURCE_DIR}/scripts/robot_gen.py
                    ${URDF_FILE}
                    ${CMAKE_SOURCE_DIR}/scripts/robot_config.cpp.template
                    ${CMAKE_CURRENT_BINARY_DIR}/forward_kinematics_test.cpp
                    ${ROOT_LINK}
                    ${TIP_LINK}`"
            VERBATIM
    )
    add_custom_target(code_generation DEPENDS forward_kinematics_lib.cpp)

    add_library(fast_forward_kinematics_library SHARED forward_kinematics_lib.cpp)
    add_dependencies(fast_forward_kinematics_library code_generation)
    add_dependencies(fast_forward_kinematics_library LBFGSpp)

    find_package(Eigen3 3.3 NO_MODULE)
    target_compile_definitions(fast_forward_kinematics_library PUBLIC "${FAST_FK_NUMBER_OF_JOINTS}")
    target_include_directories(fast_forward_kinematics_library PUBLIC ${CMAKE_SOURCE_DIR}/include)
    target_include_directories(fast_forward_kinematics_library PUBLIC ${LBFGSppIncludeDir})
    target_link_libraries(fast_forward_kinematics_library PUBLIC Eigen3::Eigen)
    target_link_libraries(fast_forward_kinematics_library PUBLIC Eigen3::Eigen)

endfunction()
```

```
40              COMMENT
41              "Running `${PYTHON_EXECUTABLE} ${CMAKE_SOURCE_DIR}/scripts/robot_gen.py
42                      ${URDF_FILE}
43                      ${CMAKE_SOURCE_DIR}/scripts/robot_config.cpp.template
44                      ${CMAKE_CURRENT_BINARY_DIR}/forward_kinematics_test.cpp
45                      ${ROOT_LINK}
46                      ${TIP_LINK}`"
47              VERBATIM
48      )
49      add_custom_target(code_generation DEPENDS forward_kinematics_lib.cpp)
50
51      add_library(fast_forward_kinematics_library SHARED forward_kinematics_lib.cpp)
52      add_dependencies(fast_forward_kinematics_library code_generation)
53      add_dependencies(fast_forward_kinematics_library LBFGSpp)
54
55      find_package(Eigen3 3.3 NO_MODULE)
56      target_compile_definitions(fast_forward_kinematics_library PUBLIC "${FAST_FK_NUMBER_OF_JOINTS}")
57      target_include_directories(fast_forward_kinematics_library PUBLIC ${CMAKE_SOURCE_DIR}/include)
58      target_include_directories(fast_forward_kinematics_library PUBLIC ${LBFGSppIncludeDir})
59      target_link_libraries(fast_forward_kinematics_library PUBLIC Eigen3::Eigen)
60      target_link_libraries(fast_forward_kinematics_library PUBLIC Eigen3::Eigen)
61
62  endfunction()
63
64
65
66
67
68
69
70
71
72
73
74
75
```

```
root@7fda3b6daf20:~/fast_robot_kinematics# cmake -Bbuild -S.
-- The C compiler identification is GNU 11.4.0
-- The CXX compiler identification is GNU 11.4.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found Python: /usr/bin/python3.10 (found version "3.10.12") found components: Interpreter
-- Python executable: /usr/bin/python3.10
'/usr/bin/python3.10' '/root/fast_robot_kinematics/scripts/get_num_joints.py' '/root/fast_robot_kinematic
s/test/urdf/robot.urdf' 'base_link' 'grasp_link'
-- Configuring done
-- Generating done
-- Build files have been written to: /root/fast_robot_kinematics/build
root@7fda3b6daf20:~/fast_robot_kinematics#
```

```
root@7fda3b6daf20:~/fast_robot_kinematics# ls build/test/
CMakeFiles  Makefile  cmake_install.cmake
root@7fda3b6daf20:~/fast_robot_kinematics#
```

```
root@7fda3b6daf20:~/fast_robot_kinematics# make -C build code_generation
```

```
root@7fda3b6daf20:~/fast_robot_kinematics# ls build/test/
```

```
-- Installing: /root/fast_robot_kinematics/build/include/LBFGSpp
-- Installing: /root/fast_robot_kinematics/build/include/LBFGSpp/BKLDLT.h
-- Installing: /root/fast_robot_kinematics/build/include/LBFGSpp/LineSearchNocedalWright.h
-- Installing: /root/fast_robot_kinematics/build/include/LBFGSpp/Cauchy.h
-- Installing: /root/fast_robot_kinematics/build/include/LBFGSpp/LineSearchMoreThuente.h
-- Installing: /root/fast_robot_kinematics/build/include/LBFGSpp/SubspaceMin.h
-- Installing: /root/fast_robot_kinematics/build/include/LBFGSpp/LineSearchBacktracking.h
-- Installing: /root/fast_robot_kinematics/build/include/LBFGSpp/BFGSMat.h
-- Installing: /root/fast_robot_kinematics/build/include/LBFGSpp/Param.h
-- Installing: /root/fast_robot_kinematics/build/include/LBFGSpp/LineSearchBracketing.h
-- Installing: /root/fast_robot_kinematics/build/include/LBFGS.h
-- Installing: /root/fast_robot_kinematics/build/lib/cmake/lbfgspp/lbfgspp-targets.cmake
-- Installing: /root/fast_robot_kinematics/build/lib/cmake/lbfgspp/lbfgspp-config.cmake
-- Installing: /root/fast_robot_kinematics/build/lib/cmake/lbfgspp/lbfgspp-config-version.cmake
make[4]: Leaving directory '/root/fast_robot_kinematics/build/LBFGSpp/src/LBFGSpp-build'
[ 69%] Completed 'LBFGSpp'
make[3]: Leaving directory '/root/fast_robot_kinematics/build'
[ 69%] Built target LBFGSpp
make[3]: Entering directory '/root/fast_robot_kinematics/build'
make[3]: Leaving directory '/root/fast_robot_kinematics/build'
make[3]: Entering directory '/root/fast_robot_kinematics/build'
[ 76%] Building CXX object test/CMakeFiles/fast_forward_kinematics_library.dir/forward_kinematics_lib.cpp
.o
[ 84%] Building CXX object test/CMakeFiles/fast_forward_kinematics_library.dir/__/src/fast_kinematics.cpp
.o
/root/fast_robot_kinematics/src/fast_kinematics.cpp:1:9: warning: #pragma once in main file
    1 | #pragma once
      |         ^~~~
```

THE CODE

```cpp
#pragma once

#include <concepts>
#include <iostream>

#include <Eigen/Core>

namespace fk_interface {

    struct IKSolverStats {
        float fx = 0;
        int niter = 0;
        float grad_norm = 0;
        bool success = false;
        std::string what;
    };

    template<typename T>
    concept KinematicInterfaceConstraint = requires(T obj, size_t ind, float value,
                                                    const Eigen::Vector<float, T::get_num_joints()> &values,
                                                    Eigen::Vector<float, T::get_num_joints()> &values_non_const,
                                                    Eigen::VectorX<float> &q_guess) {
        { T::get_num_joints() };
        { obj.set_joint(ind, value) };
        { obj.set_joints(values) };
        { obj.set_joint(ind, value, value) };
        { obj.set_joints(values, values) };
        { obj.get_joint(ind) };
        { obj.get_joints(values_non_const) };
    };


    template<typename T>
    concept ForwardKinematicInterfaceConstraint = requires(T obj, size_t ind, Eigen::Matrix<float, 4, 4> &transform) {
        { obj.get_frame(ind, transform) };
        { obj.forward_kinematics() };
```

```cpp
#pragma once

#include <concepts>
#include <iostream>

#include <Eigen/Core>

namespace fk_interface {

    struct IKSolverStats {
        float fx = 0;
        int niter = 0;
        float grad_norm = 0;
        bool success = false;
        std::string what;
    };

    template<typename T>
    concept KinematicInterfaceConstraint = requires(T obj, size_t ind, float value,
                                                    const Eigen::Vector<float, T::get_num_joints()> &values,
                                                    Eigen::Vector<float, T::get_num_joints()> &values_non_const,
                                                    Eigen::VectorX<float> &q_guess) {
        { T::get_num_joints() };
        { obj.set_joint(ind, value) };
        { obj.set_joints(values) };
        { obj.set_joint(ind, value, value) };
        { obj.set_joints(values, values) };
        { obj.get_joint(ind) };
        { obj.get_joints(values_non_const) };
    };


    template<typename T>
    concept ForwardKinematicInterfaceConstraint = requires(T obj, size_t ind, Eigen::Matrix<float, 4, 4> &transform) {
        { obj.get_frame(ind, transform) };
        { obj.forward_kinematics() };
```

```
#include <Eigen/Core>

namespace fk_interface {

    struct IKSolverStats {
        float fx = 0;
        int niter = 0;
        float grad_norm = 0;
        bool success = false;
        std::string what;
    };

    template<typename T>
    concept KinematicInterfaceConstraint = requires(T obj, size_t ind, float value,
                                                    const Eigen::Vector<float, T::get_num_joints()> &values,
                                                    Eigen::Vector<float, T::get_num_joints()> &values_non_const,
                                                    Eigen::VectorX<float> &q_guess) {
        { T::get_num_joints() };
        { obj.set_joint(ind, value) };
        { obj.set_joints(values) };
        { obj.set_joint(ind, value, value) };
        { obj.set_joints(values, values) };
        { obj.get_joint(ind) };
        { obj.get_joints(values_non_const) };
    };


    template<typename T>
    concept ForwardKinematicInterfaceConstraint = requires(T obj, size_t ind, Eigen::Matrix<float, 4, 4> &transform) {
        { obj.get_frame(ind, transform) };
        { obj.forward_kinematics() };
    };

    template<typename T>
    concept InverseInterfaceConstraint = requires(T obj, Eigen::VectorX<float> &q_guess,
                                                  Eigen::Matrix<float, 4, 4> &transform) {
        { obj.inverse_kinematics(transform, q_guess) } -> std::same_as<IKSolverStats>
    };
```

```cpp
template<typename T>
concept KinematicInterfaceConstraint = requires(T obj, size_t ind, float value,
                                                const Eigen::Vector<float, T::get_num_joints()> &values,
                                                Eigen::Vector<float, T::get_num_joints()> &values_non_const,
                                                Eigen::VectorX<float> &q_guess) {
    { T::get_num_joints() };
    { obj.set_joint(ind, value) };
    { obj.set_joints(values) };
    { obj.set_joint(ind, value, value) };
    { obj.set_joints(values, values) };
    { obj.get_joint(ind) };
    { obj.get_joints(values_non_const) };
};


template<typename T>
concept ForwardKinematicInterfaceConstraint = requires(T obj, size_t ind, Eigen::Matrix<float, 4, 4> &transform) {
    { obj.get_frame(ind, transform) };
    { obj.forward_kinematics() };
};

template<typename T>
concept InverseInterfaceConstraint = requires(T obj, Eigen::VectorX<float> &q_guess,
                                              Eigen::Matrix<float, 4, 4> &transform) {
    { obj.inverse_kinematics(transform, q_guess) } -> std::same_as<IKSolverStats>
};

template<typename KI> requires KinematicInterfaceConstraint<KI> && ForwardKinematicInterfaceConstraint<KI>
struct ForwardKinematicsInterface : KI {
};


template<typename IK> requires KinematicInterfaceConstraint<IK> && ForwardKinematicInterfaceConstraint<IK> &&
                               InverseInterfaceConstraint<IK>
struct InverseKinematicsInterface : IK {
```

```
                                                const Eigen::Vector<float, T::get_num_joints()> &values,
                                                Eigen::Vector<float, T::get_num_joints()> &values_non_const,
                                                Eigen::VectorX<float> &q_guess) {
        { T::get_num_joints() };
        { obj.set_joint(ind, value) };
        { obj.set_joints(values) };
        { obj.set_joint(ind, value, value) };
        { obj.set_joints(values, values) };
        { obj.get_joint(ind) };
        { obj.get_joints(values_non_const) };
    };


    template<typename T>
    concept ForwardKinematicInterfaceConstraint = requires(T obj, size_t ind, Eigen::Matrix<float, 4, 4> &transform) {
        { obj.get_frame(ind, transform) };
        { obj.forward_kinematics() };
    };

    template<typename T>
    concept InverseInterfaceConstraint = requires(T obj, Eigen::VectorX<float> &q_guess,
                                                  Eigen::Matrix<float, 4, 4> &transform) {
        { obj.inverse_kinematics(transform, q_guess) } -> std::same_as<IKSolverStats>
    };

    template<typename KI> requires KinematicInterfaceConstraint<KI> && ForwardKinematicInterfaceConstraint<KI>
    struct ForwardKinematicsInterface : KI {
    };


    template<typename IK> requires KinematicInterfaceConstraint<IK> && ForwardKinematicInterfaceConstraint<IK> &&
                                   InverseInterfaceConstraint<IK>
    struct InverseKinematicsInterface : IK {
    };

}
```

```
                                                const Eigen::Vector<float, T::get_num_joints()> &values,
                                                Eigen::Vector<float, T::get_num_joints()> &values_non_const,
                                                Eigen::VectorX<float> &q_guess) {
        { T::get_num_joints() };
        { obj.set_joint(ind, value) };
        { obj.set_joints(values) };
        { obj.set_joint(ind, value, value) };
        { obj.set_joints(values, values) };
        { obj.get_joint(ind) };
        { obj.get_joints(values_non_const) };
    };


    template<typename T>
    concept ForwardKinematicInterfaceConstraint = requires(T obj, size_t ind, Eigen::Matrix<float, 4, 4> &transform) {
        { obj.get_frame(ind, transform) };
        { obj.forward_kinematics() };
    };

    template<typename T>
    concept InverseInterfaceConstraint = requires(T obj, Eigen::VectorX<float> &q_guess,
                                                  Eigen::Matrix<float, 4, 4> &transform) {
        { obj.inverse_kinematics(transform, q_guess) } -> std::same_as<IKSolverStats>
    };

    template<typename KI> requires KinematicInterfaceConstraint<KI> && ForwardKinematicInterfaceConstraint<KI>
    struct ForwardKinematicsInterface : KI {
    };


    template<typename IK> requires KinematicInterfaceConstraint<IK> && ForwardKinematicInterfaceConstraint<IK> &&
                                   InverseInterfaceConstraint<IK>
    struct InverseKinematicsInterface : IK {
    };

}
```

```
20                                                  const Eigen::Vector<float, T::get_num_joints()> &values,
21                                                  Eigen::Vector<float, T::get_num_joints()> &values_non_const,
22                                                  Eigen::VectorX<float> &q_guess) {
23         { T::get_num_joints() };
24         { obj.set_joint(ind, value) };
25         { obj.set_joints(values) };
26         { obj.set_joint(ind, value, value) };
27         { obj.set_joints(values, values) };
28         { obj.get_joint(ind) };
29         { obj.get_joints(values_non_const) };
30     };
31
32
33     template<typename T>
34     concept ForwardKinematicInterfaceConstraint = requires(T obj, size_t ind, Eigen::Matrix<float, 4, 4> &transform) {
35         { obj.get_frame(ind, transform) };
36         { obj.forward_kinematics() };
37     };
38
39     template<typename T>
40     concept InverseInterfaceConstraint = requires(T obj, Eigen::VectorX<float> &q_guess,
41                                                   Eigen::Matrix<float, 4, 4> &transform) {
42         { obj.inverse_kinematics(transform, q_guess) } -> std::same_as<IKSolverStats>
43     };
44
45     template<typename KI> requires KinematicInterfaceConstraint<KI> && ForwardKinematicInterfaceConstraint<KI>
46     struct ForwardKinematicsInterface : KI {
47     };
48
49
50     template<typename IK> requires KinematicInterfaceConstraint<IK> && ForwardKinematicInterfaceConstraint<IK> &&
51                                    InverseInterfaceConstraint<IK>
52     struct InverseKinematicsInterface : IK {
53     };
54
55 }
```

```cpp
#include "chrono"
#include "iostream"

#ifdef USE_FAST_KINEMATICS

#include "fast_kinematics.hpp"
using KI = fast_fk::JointData;
#else

#include "kdl_kinematics.hpp"
using KI = kdl_impl::JointData;
#endif

int main(int arc, char **argv) {
    constexpr int iterations = 128 * 128;
    constexpr int multiplier = 128 * MULTIPLIER;
    std::array<Eigen::Vector<float, KI::get_num_joints()>, iterations> rand_values;
    for (auto &rand_val: rand_values) {
        rand_val = Eigen::Vector<float, KI::get_num_joints()>::Random();
    }

    fk_interface::ForwardKinematicsInterface<KI> fk_interface;

    auto start = std::chrono::high_resolution_clock::now();
    for (int k = 0; k < multiplier; k++) {
        for (int i = 0; i < iterations; i++) {
            fk_interface.set_joints(rand_values[i]);
            fk_interface.forward_kinematics();
        }
    }
    auto stop = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::nanoseconds>(stop - start);

    std::cout << "Time taken by function: " << (float) duration.count() << " nanoseconds" << std::endl;
    std::cout << "Average: " << ((float) duration.count()) / (iterations * multiplier) << " nanoseconds"
              << std::endl;
```

```cpp
#include "chrono"
#include "iostream"

#ifdef USE_FAST_KINEMATICS

#include "fast_kinematics.hpp"
using KI = fast_fk::JointData;
#else

#include "kdl_kinematics.hpp"
using KI = kdl_impl::JointData;
#endif

int main(int arc, char **argv) {
    constexpr int iterations = 128 * 128;
    constexpr int multiplier = 128 * MULTIPLIER;
    std::array<Eigen::Vector<float, KI::get_num_joints()>, iterations> rand_values;
    for (auto &rand_val: rand_values) {
        rand_val = Eigen::Vector<float, KI::get_num_joints()>::Random();
    }

    fk_interface::ForwardKinematicsInterface<KI> fk_interface;

    auto start = std::chrono::high_resolution_clock::now();
    for (int k = 0; k < multiplier; k++) {
        for (int i = 0; i < iterations; i++) {
            fk_interface.set_joints(rand_values[i]);
            fk_interface.forward_kinematics();
        }
    }
    auto stop = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::nanoseconds>(stop - start);

    std::cout << "Time taken by function: " << (float) duration.count() << " nanoseconds" << std::endl;
    std::cout << "Average: " << ((float) duration.count()) / (iterations * multiplier) << " nanoseconds"
              << std::endl;
```

```cpp
#include "chrono"
#include "iostream"

#ifdef USE_FAST_KINEMATICS

#include "fast_kinematics.hpp"
using KI = fast_fk::JointData;
#else

#include "kdl_kinematics.hpp"
using KI = kdl_impl::JointData;
#endif

int main(int arc, char **argv) {
    constexpr int iterations = 128 * 128;
    constexpr int multiplier = 128 * MULTIPLIER;
    std::array<Eigen::Vector<float, KI::get_num_joints()>, iterations> rand_values;
    for (auto &rand_val: rand_values) {
        rand_val = Eigen::Vector<float, KI::get_num_joints()>::Random();
    }

    fk_interface::ForwardKinematicsInterface<KI> fk_interface;

    auto start = std::chrono::high_resolution_clock::now();
    for (int k = 0; k < multiplier; k++) {
        for (int i = 0; i < iterations; i++) {
            fk_interface.set_joints(rand_values[i]);
            fk_interface.forward_kinematics();
        }
    }
    auto stop = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::nanoseconds>(stop - start);

    std::cout << "Time taken by function: " << (float) duration.count() << " nanoseconds" << std::endl;
    std::cout << "Average: " << ((float) duration.count()) / (iterations * multiplier) << " nanoseconds"
              << std::endl;
```

```cpp
#ifdef USE_FAST_KINEMATICS

#include "fast_kinematics.hpp"
using KI = fast_fk::JointData;
#else

#include "kdl_kinematics.hpp"
using KI = kdl_impl::JointData;
#endif

int main(int arc, char **argv) {
    constexpr int iterations = 128 * 128;
    constexpr int multiplier = 128 * MULTIPLIER;
    std::array<Eigen::Vector<float, KI::get_num_joints()>, iterations> rand_values;
    for (auto &rand_val: rand_values) {
        rand_val = Eigen::Vector<float, KI::get_num_joints()>::Random();
    }

    fk_interface::ForwardKinematicsInterface<KI> fk_interface;

    auto start = std::chrono::high_resolution_clock::now();
    for (int k = 0; k < multiplier; k++) {
        for (int i = 0; i < iterations; i++) {
            fk_interface.set_joints(rand_values[i]);
            fk_interface.forward_kinematics();
        }
    }
    auto stop = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::nanoseconds>(stop - start);

    std::cout << "Time taken by function: " << (float) duration.count() << " nanoseconds" << std::endl;
    std::cout << "Average: " << ((float) duration.count()) / (iterations * multiplier) << " nanoseconds"
              << std::endl;

    return 0;
}
```

```cpp
#ifdef USE_FAST_KINEMATICS

#include "fast_kinematics.hpp"
using KI = fast_fk::JointData;
#else

#include "kdl_kinematics.hpp"
using KI = kdl_impl::JointData;
#endif

int main(int arc, char **argv) {
    constexpr int iterations = 128 * 128;
    constexpr int multiplier = 128 * MULTIPLIER;
    std::array<Eigen::Vector<float, KI::get_num_joints()>, iterations> rand_values;
    for (auto &rand_val: rand_values) {
        rand_val = Eigen::Vector<float, KI::get_num_joints()>::Random();
    }

    fk_interface::ForwardKinematicsInterface<KI> fk_interface;

    auto start = std::chrono::high_resolution_clock::now();
    for (int k = 0; k < multiplier; k++) {
        for (int i = 0; i < iterations; i++) {
            fk_interface.set_joints(rand_values[i]);
            fk_interface.forward_kinematics();
        }
    }
    auto stop = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::nanoseconds>(stop - start);

    std::cout << "Time taken by function: " << (float) duration.count() << " nanoseconds" << std::endl;
    std::cout << "Average: " << ((float) duration.count()) / (iterations * multiplier) << " nanoseconds"
              << std::endl;

    return 0;
}
```

```cpp
#ifdef USE_FAST_KINEMATICS

#include "fast_kinematics.hpp"
using KI = fast_fk::JointData;
#else

#include "kdl_kinematics.hpp"
using KI = kdl_impl::JointData;
#endif

int main(int arc, char **argv) {
    constexpr int iterations = 128 * 128;
    constexpr int multiplier = 128 * MULTIPLIER;
    std::array<Eigen::Vector<float, KI::get_num_joints()>, iterations> rand_values;
    for (auto &rand_val: rand_values) {
        rand_val = Eigen::Vector<float, KI::get_num_joints()>::Random();
    }

    fk_interface::ForwardKinematicsInterface<KI> fk_interface;

    auto start = std::chrono::high_resolution_clock::now();
    for (int k = 0; k < multiplier; k++) {
        for (int i = 0; i < iterations; i++) {
            fk_interface.set_joints(rand_values[i]);
            fk_interface.forward_kinematics();
        }
    }
    auto stop = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::nanoseconds>(stop - start);

    std::cout << "Time taken by function: " << (float) duration.count() << " nanoseconds" << std::endl;
    std::cout << "Average: " << ((float) duration.count()) / (iterations * multiplier) << " nanoseconds"
              << std::endl;

    return 0;
}
```

```cpp
#include "chrono"
#include "iostream"

#ifdef USE_FAST_KINEMATICS

#include "fast_kinematics.hpp"
using IK = fast_fk::JointData;
#else

#include "kdl_kinematics.hpp"
using IK = kdl_impl::JointData;
#endif

int main(int arc, char **argv) {
    constexpr int iterations = 128 * 128 * 5 * MULTIPLIER;

    // get target pose
    Eigen::Matrix<float, 4, 4> tf;
    Eigen::VectorX<float> q_in = Eigen::VectorX<float>::Random(IK::get_num_joints());
    fk_interface::InverseKinematicsInterface<IK> fk_interface;
    fk_interface.set_joints(q_in);
    fk_interface.forward_kinematics();
    fk_interface.get_frame(IK::get_num_joints() - 1, tf);

    auto start = std::chrono::high_resolution_clock::now();
    fk_interface::IKSolverStats stats;
    size_t failed = 0;
    size_t succeeded = 0;
    Eigen::VectorX<float> q;
    for (int ind = 0; ind < iterations; ++ind) {
        q = Eigen::VectorX<float>::Random(IK::get_num_joints());
        stats = fk_interface.inverse_kinematics(tf, q);
        failed += stats.success == 0;
        succeeded += stats.success == 1;
    }
    auto stop = std::chrono::high_resolution_clock::now();
```

```cpp
#include "chrono"
#include "iostream"

#ifdef USE_FAST_KINEMATICS

#include "fast_kinematics.hpp"
using IK = fast_fk::JointData;
#else

#include "kdl_kinematics.hpp"
using IK = kdl_impl::JointData;
#endif

int main(int arc, char **argv) {
    constexpr int iterations = 128 * 128 * 5 * MULTIPLIER;

    // get target pose
    Eigen::Matrix<float, 4, 4> tf;
    Eigen::VectorX<float> q_in = Eigen::VectorX<float>::Random(IK::get_num_joints());
    fk_interface::InverseKinematicsInterface<IK> fk_interface;
    fk_interface.set_joints(q_in);
    fk_interface.forward_kinematics();
    fk_interface.get_frame(IK::get_num_joints() - 1, tf);

    auto start = std::chrono::high_resolution_clock::now();
    fk_interface::IKSolverStats stats;
    size_t failed = 0;
    size_t succeeded = 0;
    Eigen::VectorX<float> q;
    for (int ind = 0; ind < iterations; ++ind) {
        q = Eigen::VectorX<float>::Random(IK::get_num_joints());
        stats = fk_interface.inverse_kinematics(tf, q);
        failed += stats.success == 0;
        succeeded += stats.success == 1;
    }
    auto stop = std::chrono::high_resolution_clock::now();
```

```cpp
#include <iostream>

#ifdef USE_FAST_KINEMATICS

#include "fast_kinematics.hpp"
using IK = fast_fk::JointData;
#else

#include "kdl_kinematics.hpp"
using IK = kdl_impl::JointData;
#endif

int main(int arc, char **argv) {
    constexpr int iterations = 128 * 128 * 5 * MULTIPLIER;

    // get target pose
    Eigen::Matrix<float, 4, 4> tf;
    Eigen::VectorX<float> q_in = Eigen::VectorX<float>::Random(IK::get_num_joints());
    fk_interface::InverseKinematicsInterface<IK> fk_interface;
    fk_interface.set_joints(q_in);
    fk_interface.forward_kinematics();
    fk_interface.get_frame(IK::get_num_joints() - 1, tf);

    auto start = std::chrono::high_resolution_clock::now();
    fk_interface::IKSolverStats stats;
    size_t failed = 0;
    size_t succeeded = 0;
    Eigen::VectorX<float> q;
    for (int ind = 0; ind < iterations; ++ind) {
        q = Eigen::VectorX<float>::Random(IK::get_num_joints());
        stats = fk_interface.inverse_kinematics(tf, q);
        failed += stats.success == 0;
        succeeded += stats.success == 1;
    }
    auto stop = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::microseconds>(stop - start);
```

```cpp
using IK = kdl_impl::JointData;
#endif

int main(int arc, char **argv) {
    constexpr int iterations = 128 * 128 * 5 * MULTIPLIER;

    // get target pose
    Eigen::Matrix<float, 4, 4> tf;
    Eigen::VectorX<float> q_in = Eigen::VectorX<float>::Random(IK::get_num_joints());
    fk_interface::InverseKinematicsInterface<IK> fk_interface;
    fk_interface.set_joints(q_in);
    fk_interface.forward_kinematics();
    fk_interface.get_frame(IK::get_num_joints() - 1, tf);

    auto start = std::chrono::high_resolution_clock::now();
    fk_interface::IKSolverStats stats;
    size_t failed = 0;
    size_t succeeded = 0;
    Eigen::VectorX<float> q;
    for (int ind = 0; ind < iterations; ++ind) {
        q = Eigen::VectorX<float>::Random(IK::get_num_joints());
        stats = fk_interface.inverse_kinematics(tf, q);
        failed += stats.success == 0;
        succeeded += stats.success == 1;
    }
    auto stop = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::microseconds>(stop - start);

    std::cout << "Time taken by function: " << (double) duration.count() << " microseconds" << std::endl;
    std::cout << "Average: " << ((double) duration.count()) / (iterations) << " microseconds"
              << std::endl;
    std::cout << "Percent success: " << ((double) (100.0 * succeeded) / (failed + succeeded)) << "%"
              << std::endl;

    return 0;
}
```

```cpp
using IK = kdl_impl::JointData;
#endif

int main(int arc, char **argv) {
    constexpr int iterations = 128 * 128 * 5 * MULTIPLIER;

    // get target pose
    Eigen::Matrix<float, 4, 4> tf;
    Eigen::VectorX<float> q_in = Eigen::VectorX<float>::Random(IK::get_num_joints());
    fk_interface::InverseKinematicsInterface<IK> fk_interface;
    fk_interface.set_joints(q_in);
    fk_interface.forward_kinematics();
    fk_interface.get_frame(IK::get_num_joints() - 1, tf);

    auto start = std::chrono::high_resolution_clock::now();
    fk_interface::IKSolverStats stats;
    size_t failed = 0;
    size_t succeeded = 0;
    Eigen::VectorX<float> q;
    for (int ind = 0; ind < iterations; ++ind) {
        q = Eigen::VectorX<float>::Random(IK::get_num_joints());
        stats = fk_interface.inverse_kinematics(tf, q);
        failed += stats.success == 0;
        succeeded += stats.success == 1;
    }
    auto stop = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::microseconds>(stop - start);

    std::cout << "Time taken by function: " << (double) duration.count() << " microseconds" << std::endl;
    std::cout << "Average: " << ((double) duration.count()) / (iterations) << " microseconds"
              << std::endl;
    std::cout << "Percent success: " << ((double) (100.0 * succeeded) / (failed + succeeded)) << "%"
              << std::endl;

    return 0;
}
```

```cpp
#include "fast_kinematics.hpp"

namespace fast_fk::internal {
    // sin(t) cos(t)  px py pz R11, R12, R13...
    constexpr int data_size = 2 + 3 + 9 + 2; // 16

    void forward_kinematics_internal(float *input_data, std::size_t /*size*/) {

        // row major
        constexpr std::array<float, 6 * 9> R_all = { -1.0, -0.0, 0.0,
                                                     0.0, -1.0, 0.0,
                                                     0.0, 0.0, 1.0,
                                                     0.9999999999920636, -0.0, 0.0,
                                                     -0.0, -0.0007013484622411433, 0.9999997540472005,
                                                     0.0, -0.999999754055137, -0.0007013484622467095,
                                                     0.9999993163649181, 0.0, -0.001169289425224364,
                                                     -0.0, 0.9999994508798468, 0.0010479601216713834,
                                                     0.001169294705420077, -0.00104795423012248, 0.9999987672701519,
                                                     0.9999999072555382, -1.0253697707014652e-05, 0.0004305621635032162
                                                     0.0, 0.9999720394582239, 0.007477984507041285,
                                                     -0.0004306268017444494, -0.007477980785065483, 0.9999719467884766,
                                                     0.9999999999999949, -0.0, -0.0,
                                                     0.0, 0.001812660647159262, 0.9999983571293345,
                                                     0.0, -0.9999983571293396, 0.001812660647159271,
                                                     0.9999999999999998, 0.0, -0.0,
                                                     -0.0, 0.0011436177325240319, -0.9999993460690269,
                                                     -0.0, 0.9999993460690271, 0.001143617732524032,
        };

        constexpr std::array<float, 6 * 3> offset_all = { 0.0, 0.0, 0.1625702965797758,
                                                          0.000182214465989093, 0.0, 0.0,
                                                          -0.4249817627044961, 0.0, 0.0,
                                                          -0.3921666446509172, -0.0009975307642066673, 0.133391995638352
                                                          0.0, -0.09959821611958637, 0.0001805380634879481,
                                                          7.603964784130673e-05, 0.09950302422228456, 0.000113793497353
        };
```

```cpp
#include "fast_kinematics.hpp"

namespace fast_fk::internal {
    // sin(t) cos(t)  px py pz R11, R12, R13...
    constexpr int data_size = 2 + 3 + 9 + 2; // 16

    void forward_kinematics_internal(float *input_data, std::size_t /*size*/) {

        // row major
        constexpr std::array<float, 6 * 9> R_all = { -1.0, -0.0, 0.0,
                                                     0.0, -1.0, 0.0,
                                                     0.0, 0.0, 1.0,
                                                     0.9999999999920636, -0.0, 0.0,
                                                     -0.0, -0.0007013484622411433, 0.9999997540472005,
                                                     0.0, -0.999999754055137, -0.0007013484622467095,
                                                     0.9999993163649181, 0.0, -0.001169289425224364,
                                                     -0.0, 0.9999994508798468, 0.0010479601216713834,
                                                     0.001169294705420077, -0.00104795423012248, 0.9999987672701519,
                                                     0.9999999072555382, -1.0253697707014652e-05, 0.0004305621635032162
                                                     0.0, 0.9999720394582239, 0.007477984507041285,
                                                     -0.0004306268017444494, -0.007477980785065483, 0.9999719467884766,
                                                     0.9999999999999949, -0.0, -0.0,
                                                     0.0, 0.001812660647159262, 0.9999983571293345,
                                                     0.0, -0.9999983571293396, 0.001812660647159271,
                                                     0.9999999999999998, 0.0, -0.0,
                                                     -0.0, 0.0011436177325240319, -0.9999993460690269,
                                                     -0.0, 0.9999993460690271, 0.001143617732524032,
        };

        constexpr std::array<float, 6 * 3> offset_all = { 0.0, 0.0, 0.1625702965797758,
                                                          0.000182214465989093, 0.0, 0.0,
                                                          -0.4249817627044961, 0.0, 0.0,
                                                          -0.3921666446509172, -0.000997530764206673, 0.133391995638352
                                                          0.0, -0.09959821611958637, 0.0001805380634879481,
                                                          7.603964784130673e-05, 0.09950302422228456, 0.000113793497353
        };
```

```
    constexpr int data_size = 2 + 3 + 9 + 2; // 16

    void forward_kinematics_internal(float *input_data, std::size_t /*size*/) {

        // row major
        constexpr std::array<float, 6 * 9> R_all = { -1.0, -0.0, 0.0,
                                                     0.0, -1.0, 0.0,
                                                     0.0, 0.0, 1.0,
                                                     0.9999999999920636, -0.0, 0.0,
                                                     -0.0, -0.0007013484622411433, 0.9999997540472005,
                                                     0.0, -0.999999754055137, -0.0007013484622467095,
                                                     0.9999993163649181, 0.0, -0.001169289425224364,
                                                     -0.0, 0.9999994508798468, 0.0010479601216713834,
                                                     0.001169294705420077, -0.00104795423012248, 0.9999987672701519,
                                                     0.9999999072555382, -1.0253697707014652e-05, 0.0004305621635032162
                                                     0.0, 0.9999720394582239, 0.007477984507041285,
                                                     -0.0004306268017444494, -0.007477980785065483, 0.9999719467884766,
                                                     0.9999999999999949, -0.0, -0.0,
                                                     0.0, 0.001812660647159262, 0.9999983571293345,
                                                     0.0, -0.9999983571293396, 0.0018126606471592713,
                                                     0.9999999999999998, 0.0, -0.0,
                                                     -0.0, 0.0011436177325240319, -0.9999993460690269,
                                                     -0.0, 0.9999993460690271, 0.001143617732524032,
        };

        constexpr std::array<float, 6 * 3> offset_all = { 0.0, 0.0, 0.1625702965797758,
                                                          0.0001822144659890 93, 0.0, 0.0,
                                                          -0.4249817627044961, 0.0, 0.0,
                                                          -0.3921666446509172, -0.0009975307642066673, 0.133391995638352
                                                          0.0, -0.09959821611958637, 0.0001805380634879481,
                                                          7.603964784130673e-05, 0.09950302422228456, 0.000113793497353
        };

        float tmp1;
        float tmp2;
        float tmp3;
```

```
                                                  -0.0004500260017444494, -0.007477980785065485, 0.9999719407884706,
                                                  0.9999999999999949, -0.0, -0.0,
                                                  0.0, 0.001812660647159262, 0.9999983571293345,
                                                  0.0, -0.9999983571293396, 0.001812660647159262,
                                                  0.9999999999999998, 0.0, -0.0,
                                                  -0.0, 0.0011436177325240319, -0.9999993460690269,
                                                  -0.0, 0.9999993460690271, 0.001143617732524032,
    };

    constexpr std::array<float, 6 * 3> offset_all = { 0.0, 0.0, 0.1625702965797758,
                                                      0.000182214465989093, 0.0, 0.0,
                                                      -0.4249817627044961, 0.0, 0.0,
                                                      -0.3921666446509172, -0.0009975307642066673, 0.133391995638352
                                                      0.0, -0.09959821611958637, 0.0001805380634879481,
                                                      7.603964784130673e-05, 0.09950302422228456, 0.0001137934973534
    };

    float tmp1;
    float tmp2;
    float tmp3;


    {
        constexpr std::size_t ind = 0;

        // R_fixed is the rotation from joint_i+1 in frame joint_i (row major)
        const float &R11_fixed = R_all[ind * 9 + 0];
        const float &R21_fixed = R_all[ind * 9 + 1];
        const float &R31_fixed = R_all[ind * 9 + 2];
        const float &R12_fixed = R_all[ind * 9 + 3];
        const float &R22_fixed = R_all[ind * 9 + 4];
        const float &R32_fixed = R_all[ind * 9 + 5];
        const float &R13_fixed = R_all[ind * 9 + 6];
        const float &R23_fixed = R_all[ind * 9 + 7];
        const float &R33_fixed = R_all[ind * 9 + 8];

```

```
42
43      {
44          constexpr std::size_t ind = 0;
45
46          // R_fixed is the rotation from joint_i+1 in frame joint_i (row major)
47          const float &R11_fixed = R_all[ind * 9 + 0];
48          const float &R21_fixed = R_all[ind * 9 + 1];
49          const float &R31_fixed = R_all[ind * 9 + 2];
50          const float &R12_fixed = R_all[ind * 9 + 3];
51          const float &R22_fixed = R_all[ind * 9 + 4];
52          const float &R32_fixed = R_all[ind * 9 + 5];
53          const float &R13_fixed = R_all[ind * 9 + 6];
54          const float &R23_fixed = R_all[ind * 9 + 7];
55          const float &R33_fixed = R_all[ind * 9 + 8];
56
57          // offset is the offset of joint_i+1 in frame joint_i
58          const float &offset_x = offset_all[ind * 3 + 0];
59          const float &offset_y = offset_all[ind * 3 + 1];
60          const float &offset_z = offset_all[ind * 3 + 2];
61
62          // R is the rotation from joint_i+1 in the base frame (row major)
63          float &R11 = input_data[ind * data_size + 5];
64          float &R12 = input_data[ind * data_size + 6];
65          float &R13 = input_data[ind * data_size + 7];
66          float &R21 = input_data[ind * data_size + 8];
67          float &R22 = input_data[ind * data_size + 9];
68          float &R23 = input_data[ind * data_size + 10];
69          float &R31 = input_data[ind * data_size + 11];
70          float &R32 = input_data[ind * data_size + 12];
71          float &R33 = input_data[ind * data_size + 13];
72
73
74          const float &sin_theta = input_data[ind * data_size + 0];
75          const float &cos_theta = input_data[ind * data_size + 1];
76
77          // apply revolute rotation to R_fixed and store in joint_to_base_rotation_R
```

```
            float &R21 = input_data[ind * data_size + 8];
            float &R22 = input_data[ind * data_size + 9];
            float &R23 = input_data[ind * data_size + 10];
            float &R31 = input_data[ind * data_size + 11];
            float &R32 = input_data[ind * data_size + 12];
            float &R33 = input_data[ind * data_size + 13];


            const float &sin_theta = input_data[ind * data_size + 0];
            const float &cos_theta = input_data[ind * data_size + 1];

            // apply revolute rotation to R_fixed and store in joint to base rotation R
            R11 = R11_fixed * cos_theta + R12_fixed * sin_theta;
            R12 = -R11_fixed * sin_theta + R12_fixed * cos_theta;
            R13 = R13_fixed;
            R21 = R21_fixed * cos_theta + R22_fixed * sin_theta;
            R22 = -R21_fixed * sin_theta + R22_fixed * cos_theta;
            R23 = R23_fixed;
            R31 = R31_fixed * cos_theta + R32_fixed * sin_theta;
            R32 = -R31_fixed * sin_theta + R32_fixed * cos_theta;
            R33 = R33_fixed;

            // rotate offset to be in base frame
            input_data[ind * data_size + 2] = offset_x;
            input_data[ind * data_size + 3] = offset_y;
            input_data[ind * data_size + 4] = offset_z;




        }

        {
            constexpr std::size_t ind = 1;

            // R_fixed is the rotation from joint i+1 in frame joint i (row major)
```

```
97
98          {
99              constexpr std::size_t ind = 1;
100
101             // R_fixed is the rotation from joint_i+1 in frame joint_i (row major)
102             const float &R11_fixed = R_all[ind * 9 + 0];
103             const float &R21_fixed = R_all[ind * 9 + 1];
104             const float &R31_fixed = R_all[ind * 9 + 2];
105             const float &R12_fixed = R_all[ind * 9 + 3];
106             const float &R22_fixed = R_all[ind * 9 + 4];
107             const float &R32_fixed = R_all[ind * 9 + 5];
108             const float &R13_fixed = R_all[ind * 9 + 6];
109             const float &R23_fixed = R_all[ind * 9 + 7];
110             const float &R33_fixed = R_all[ind * 9 + 8];
111
112             // offset is the offset of joint_i+1 in frame joint_i
113             const float &offset_x = offset_all[ind * 3 + 0];
114             const float &offset_y = offset_all[ind * 3 + 1];
115             const float &offset_z = offset_all[ind * 3 + 2];
116
117             // R is the rotation from joint_i+1 in the base frame (row major)
118             float &R11 = input_data[ind * data_size + 5];
119             float &R12 = input_data[ind * data_size + 6];
120             float &R13 = input_data[ind * data_size + 7];
121             float &R21 = input_data[ind * data_size + 8];
122             float &R22 = input_data[ind * data_size + 9];
123             float &R23 = input_data[ind * data_size + 10];
124             float &R31 = input_data[ind * data_size + 11];
125             float &R32 = input_data[ind * data_size + 12];
126             float &R33 = input_data[ind * data_size + 13];
127
128
129             const float &sin_theta = input_data[ind * data_size + 0];
130             const float &cos_theta = input_data[ind * data_size + 1];
131
132             // R_old is the rotation from joint_i in base frame (column major)
```

```
130         const float &cos_theta = input_data[ind * data_size + 1];
131
132         // R_old is the rotation from joint_i in base frame (column major)
133         float &R11_old = input_data[(ind - 1) * data_size + 5];
134         float &R12_old = input_data[(ind - 1) * data_size + 6];
135         float &R13_old = input_data[(ind - 1) * data_size + 7];
136         float &R21_old = input_data[(ind - 1) * data_size + 8];
137         float &R22_old = input_data[(ind - 1) * data_size + 9];
138         float &R23_old = input_data[(ind - 1) * data_size + 10];
139         float &R31_old = input_data[(ind - 1) * data_size + 11];
140         float &R32_old = input_data[(ind - 1) * data_size + 12];
141         float &R33_old = input_data[(ind - 1) * data_size + 13];
142
143         // R_tmp is the rotation from joint_i+1 in frame joint_i after rotation applied (column major)
144         float &R11_tmp = input_data[ind * data_size + 5];
145         float &R12_tmp = input_data[ind * data_size + 6];
146         float &R13_tmp = input_data[ind * data_size + 7];
147         float &R21_tmp = input_data[ind * data_size + 8];
148         float &R22_tmp = input_data[ind * data_size + 9];
149         float &R23_tmp = input_data[ind * data_size + 10];
150         float &R31_tmp = input_data[ind * data_size + 11];
151         float &R32_tmp = input_data[ind * data_size + 12];
152         float &R33_tmp = input_data[ind * data_size + 13];
153
154         // apply revolute rotation to R11_fixed and store in R11_tmp
155         R11_tmp = R11_fixed * cos_theta + R12_fixed * sin_theta;
156         R12_tmp = -R11_fixed * sin_theta + R12_fixed * cos_theta;
157         R13_tmp = R13_fixed;
158         R21_tmp = R21_fixed * cos_theta + R22_fixed * sin_theta;
159         R22_tmp = -R21_fixed * sin_theta + R22_fixed * cos_theta;
160         R23_tmp = R23_fixed;
161         R31_tmp = R31_fixed * cos_theta + R32_fixed * sin_theta;
162         R32_tmp = -R31_fixed * sin_theta + R32_fixed * cos_theta;
163         R33_tmp = R33_fixed;
164
165         // apply R11_old rotation R11_tmp  to make it in base frame
```

```
163             R33_tmp = R33_fixed;
164
165             // apply R11_old rotation R11_tmp, to make it in base frame
166             tmp1 = R11_old * R11_tmp + R12_old * R21_tmp + R13_old * R31_tmp;
167             tmp2 = R21_old * R11_tmp + R22_old * R21_tmp + R23_old * R31_tmp;
168             tmp3 = R31_old * R11_tmp + R32_old * R21_tmp + R33_old * R31_tmp;
169             R11 = tmp1;
170             R21 = tmp2;
171             R31 = tmp3;
172             tmp1 = R11_old * R12_tmp + R12_old * R22_tmp + R13_old * R32_tmp;
173             tmp2 = R21_old * R12_tmp + R22_old * R22_tmp + R23_old * R32_tmp;
174             tmp3 = R31_old * R12_tmp + R32_old * R22_tmp + R33_old * R32_tmp;
175             R12 = tmp1;
176             R22 = tmp2;
177             R32 = tmp3;
178             tmp1 = R11_old * R13_tmp + R12_old * R23_tmp + R13_old * R33_tmp;
179             tmp2 = R21_old * R13_tmp + R22_old * R23_tmp + R23_old * R33_tmp;
180             tmp3 = R31_old * R13_tmp + R32_old * R23_tmp + R33_old * R33_tmp;
181             R13 = tmp1;
182             R23 = tmp2;
183             R33 = tmp3;
184
185             // p_old is the position of joint_i in base frame
186             const float &px_old = input_data[(ind - 1) * data_size + 2];
187             const float &py_old = input_data[(ind - 1) * data_size + 3];
188             const float &pz_old = input_data[(ind - 1) * data_size + 4];
189
190             // rotate offset to be in base frame and add p_old
191             tmp1 = R11_old * offset_x + R12_old * offset_y + R13_old * offset_z + px_old;
192             tmp2 = R21_old * offset_x + R22_old * offset_y + R23_old * offset_z + py_old;
193             tmp3 = R31_old * offset_x + R32_old * offset_y + R33_old * offset_z + pz_old;
194             input_data[ind * data_size + 2] = tmp1;
195             input_data[ind * data_size + 3] = tmp2;
196             input_data[ind * data_size + 4] = tmp3;
197
```

```
610             input_data[ind * data_size + 2] = tmp1;
611             input_data[ind * data_size + 3] = tmp2;
612             input_data[ind * data_size + 4] = tmp3;
613
614
615
616         }
617
618     }
619
620 }
621
622
623 namespace fast_fk::internal {
624     constexpr float axis_scale = 1;
625
626     float InverseKinematics::operator()(const Eigen::VectorX<float> &q, Eigen::VectorX<float> &grad) {
627         // construct input_data from q
628         float tmp1;
629         float tmp2;
630
631         tmp1 = q[0];
632         tmp2 = q[0];
633         joint_data[0][0] = std::sin(tmp1);
634         joint_data[0][1] = std::cos(tmp2);
635         tmp1 = q[1];
636         tmp2 = q[1];
637         joint_data[1][0] = std::sin(tmp1);
638         joint_data[1][1] = std::cos(tmp2);
639         tmp1 = q[2];
640         tmp2 = q[2];
641         joint_data[2][0] = std::sin(tmp1);
642         joint_data[2][1] = std::cos(tmp2);
643         tmp1 = q[3];
644         tmp2 = q[3];
645         joint_data[3][0] = std::sin(tmp1);
```

```
626 float InverseKinematics::operator()(const Eigen::VectorXf &q, Eigen::VectorXf &grad) {
627     // construct input_data from q
628     float tmp1;
629     float tmp2;
630
631     tmp1 = q[0];
632     tmp2 = q[0];
633     joint_data[0][0] = std::sin(tmp1);
634     joint_data[0][1] = std::cos(tmp2);
635     tmp1 = q[1];
636     tmp2 = q[1];
637     joint_data[1][0] = std::sin(tmp1);
638     joint_data[1][1] = std::cos(tmp2);
639     tmp1 = q[2];
640     tmp2 = q[2];
641     joint_data[2][0] = std::sin(tmp1);
642     joint_data[2][1] = std::cos(tmp2);
643     tmp1 = q[3];
644     tmp2 = q[3];
645     joint_data[3][0] = std::sin(tmp1);
646     joint_data[3][1] = std::cos(tmp2);
647     tmp1 = q[4];
648     tmp2 = q[4];
649     joint_data[4][0] = std::sin(tmp1);
650     joint_data[4][1] = std::cos(tmp2);
651     tmp1 = q[5];
652     tmp2 = q[5];
653     joint_data[5][0] = std::sin(tmp1);
654     joint_data[5][1] = std::cos(tmp2);
655
656     float *input_data = joint_data.data()->data();
657     internal::forward_kinematics_internal(input_data, data_size * 6);
658
659     Eigen::Vector3<float> target_x_axis_ = target_rot_.block<3, 1>(0, 0);
660     Eigen::Vector3<float> target_y_axis_ = target_rot_.block<3, 1>(0, 1);
661     Eigen::Vector3<float> target_z_axis_ = target_rot_.block<3, 1>(0, 2);
```

```
        Eigen::Vector3<float> target_y_axis_ = target_rot_.block<3, 1>(0, 1);
        Eigen::Vector3<float> target_z_axis_ = target_rot_.block<3, 1>(0, 2);
        Eigen::Vector3<float> target_x_ = target_pose_ + axis_scale * target_x_axis_;
        Eigen::Vector3<float> target_y_ = target_pose_ + axis_scale * target_y_axis_;
        Eigen::Vector3<float> target_z_ = target_pose_ + axis_scale * target_z_axis_;


        Eigen::Vector<float, 3> ee_pose;
        ee_pose << input_data[(6 - 1) *data_size + 2], input_data[(6 - 1)
                                                                  *data_size + 3], input_data[(6 - 1) *data_size + 4];

        // x point
        Eigen::Vector<float, 3> x_axis;
        x_axis << input_data[(6 - 1) *data_size + 5], input_data[(6 - 1)
                                                                 *data_size + 8], input_data[(6 - 1) *data_size + 11];
        Eigen::Vector<float, 3> current_x = ee_pose + x_axis;

        // y point
        Eigen::Vector<float, 3> y_axis;
        y_axis << input_data[(6 - 1) *data_size + 6], input_data[(6 - 1)
                                                                 *data_size + 9], input_data[(6 - 1) *data_size + 12];
        Eigen::Vector<float, 3> current_y = ee_pose + y_axis;

        // z point
        Eigen::Vector<float, 3> z_axis;
        z_axis << input_data[(6 - 1) *data_size + 7], input_data[(6 - 1)
                                                                 *data_size + 10], input_data[(6 - 1) *data_size + 13];
        Eigen::Vector<float, 3> current_z = ee_pose + z_axis;

        float fx = 0.0;
        Eigen::Vector3<float> delta_x = target_x_.array() - current_x.array();
        Eigen::Vector3<float> delta_y = target_y_.array() - current_y.array();
        Eigen::Vector3<float> delta_z = target_z_.array() - current_z.array();
        fx += (delta_x.array() * delta_x.array()).sum();
        fx += (delta_y.array() * delta_y.array()).sum();
```

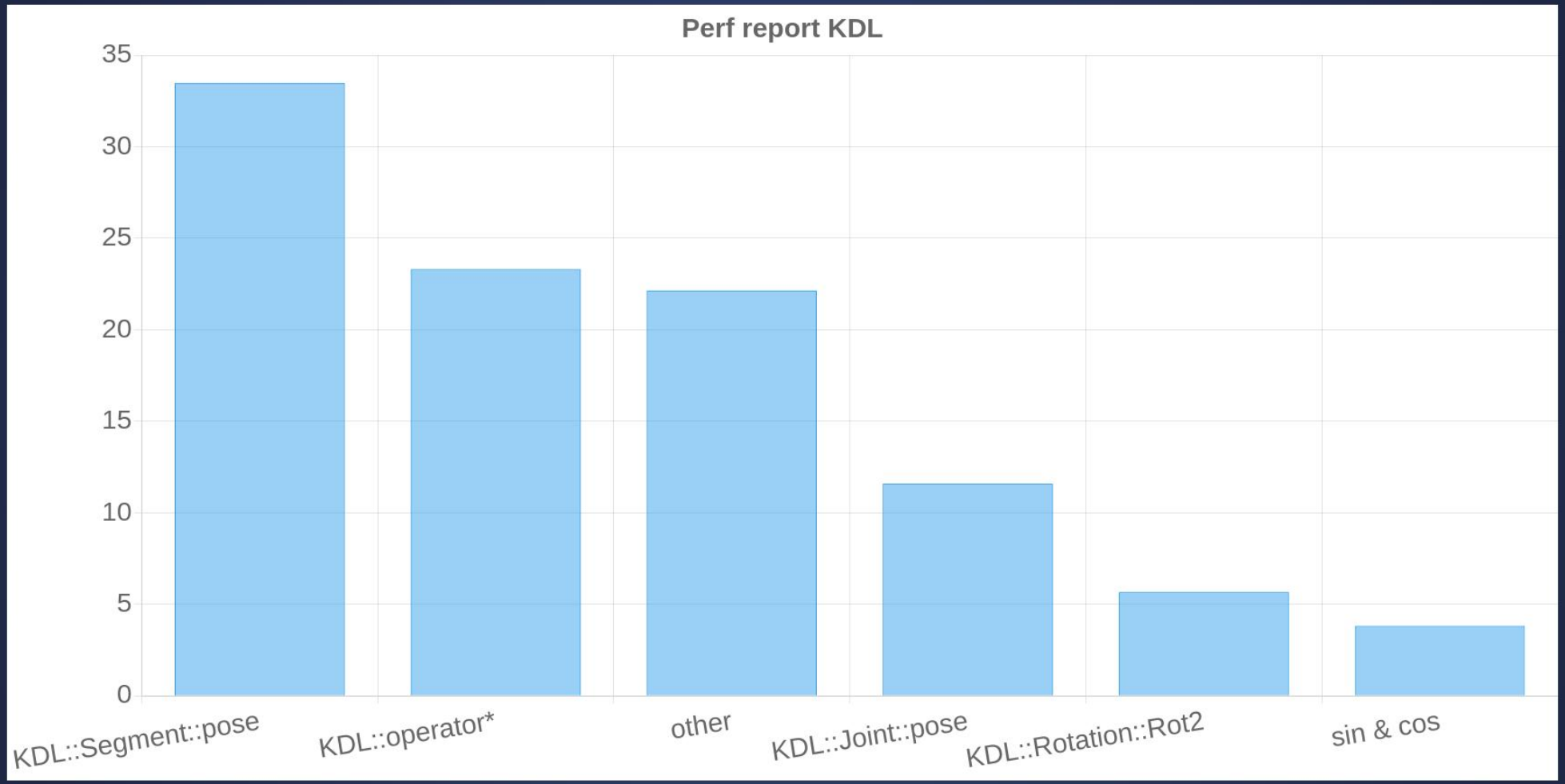
```
                                                                    *data_size + 8], input_data[(6 - 1) *data_size + 11];
    Eigen::Vector<float, 3> current_x = ee_pose + x_axis;

    // y point
    Eigen::Vector<float, 3> y_axis;
    y_axis << input_data[(6 - 1) *data_size + 6], input_data[(6 - 1)
                                                                    *data_size + 9], input_data[(6 - 1) *data_size + 12];
    Eigen::Vector<float, 3> current_y = ee_pose + y_axis;

    // z point
    Eigen::Vector<float, 3> z_axis;
    z_axis << input_data[(6 - 1) *data_size + 7], input_data[(6 - 1)
                                                                    *data_size + 10], input_data[(6 - 1) *data_size + 13];
    Eigen::Vector<float, 3> current_z = ee_pose + z_axis;

    float fx = 0.0;
    Eigen::Vector3<float> delta_x = target_x_.array() - current_x.array();
    Eigen::Vector3<float> delta_y = target_y_.array() - current_y.array();
    Eigen::Vector3<float> delta_z = target_z_.array() - current_z.array();
    fx += (delta_x.array() * delta_x.array()).sum();
    fx += (delta_y.array() * delta_y.array()).sum();
    fx += (delta_z.array() * delta_z.array()).sum();

    // J =  (delta_x)^2
    //    + (delta_y)^2
    //    + (delta_z)^2

    // dJ/dq = dJ1/dq + dJ2/dq + dJ3/dq

    // dJ1/dq = dJ/ddelta_x * ddelta_x/dq
    // dJ2/dq = dJ/ddelta_y * ddelta_y/dq
    // dJ3/dq = dJ/ddelta_z * ddelta_z/dq

    // dJ/ddelta_x : 1x3 2*(x_target - x_current)
    // dJ/ddelta_y : 1x3 2*(y_target - y_current)
    // dJ/ddelta_z : 1x3 2*(z_target - z_current)
```

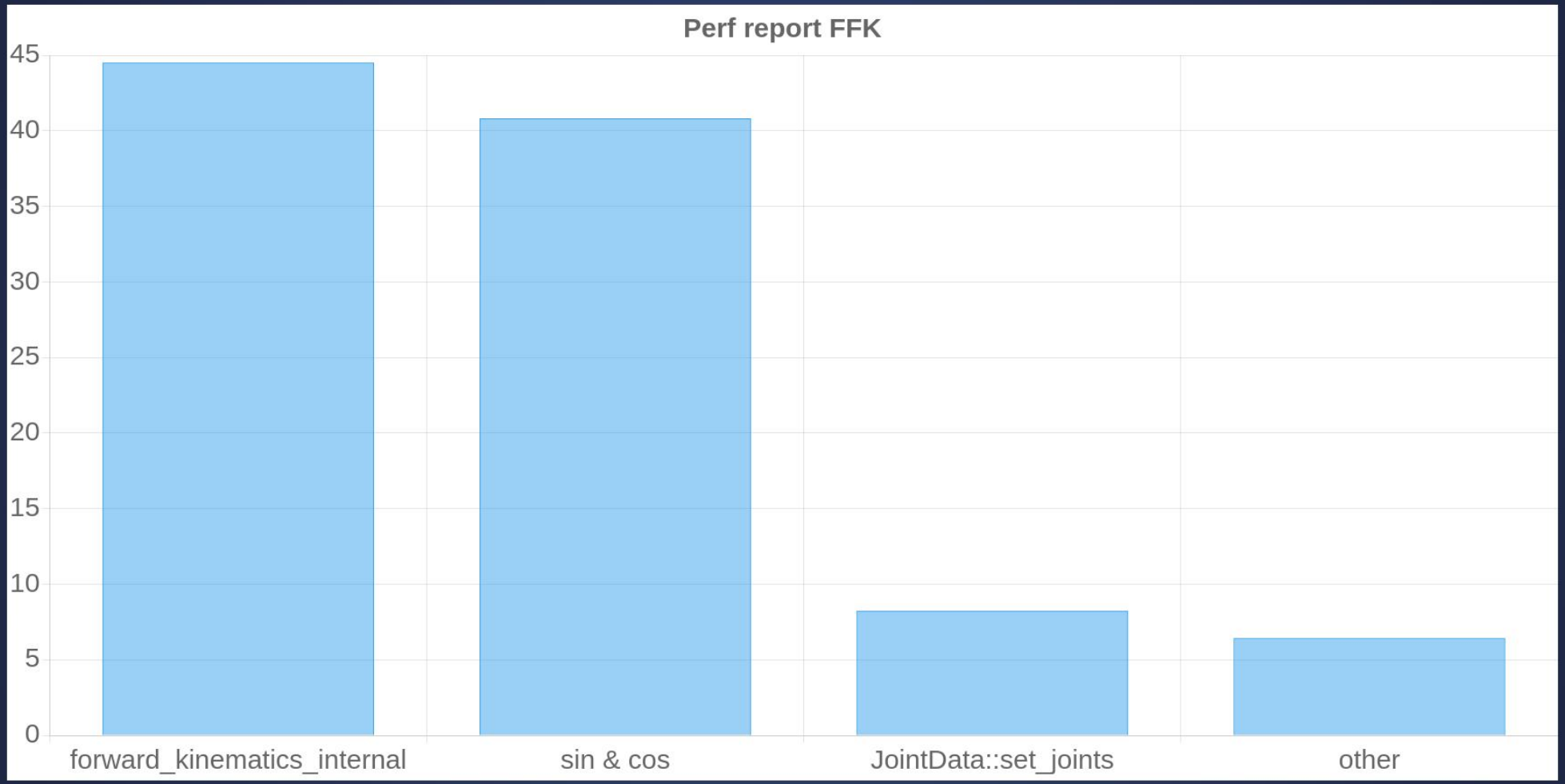
```
        Eigen::Vector<float, 3> current_z = ee_pose + z_axis;

        float fx = 0.0;
        Eigen::Vector3<float> delta_x = target_x_.array() - current_x.array();
        Eigen::Vector3<float> delta_y = target_y_.array() - current_y.array();
        Eigen::Vector3<float> delta_z = target_z_.array() - current_z.array();
        fx += (delta_x.array() * delta_x.array()).sum();
        fx += (delta_y.array() * delta_y.array()).sum();
        fx += (delta_z.array() * delta_z.array()).sum();

        // J =  (delta_x)^2
        //    + (delta_y)^2
        //    + (delta_z)^2

        // dJ/dq = dJ1/dq + dJ2/dq + dJ3/dq

        // dJ1/dq = dJ/ddelta_x * ddelta_x/dq
        // dJ2/dq = dJ/ddelta_y * ddelta_y/dq
        // dJ3/dq = dJ/ddelta_z * ddelta_z/dq

        // dJ/ddelta_x : 1x3 2*(x_target - x_current)
        // dJ/ddelta_y : 1x3 2*(y_target - y_current)
        // dJ/ddelta_z : 1x3 2*(z_target - z_current)

        // ddelta_x/dq: 3 x dofs J_{:3,:} + dofs J_{3:,:} x x_axis
        // ddelta_y/dq: 3 x dofs J_{:3,:} + dofs J_{3:,:} x y_axis
        // ddelta_z/dq: 3 x dofs J_{:3,:} + dofs J_{3:,:} x z_axis


        // gradient
        grad *= 0;
        Eigen::Vector3<float> dJ_ddelta_x = 2 * delta_x;
        Eigen::Vector3<float> dJ_ddelta_y = 2 * delta_y;
        Eigen::Vector3<float> dJ_ddelta_z = 2 * delta_z;

        Eigen::Vector<float, 3> joint_pose;
```

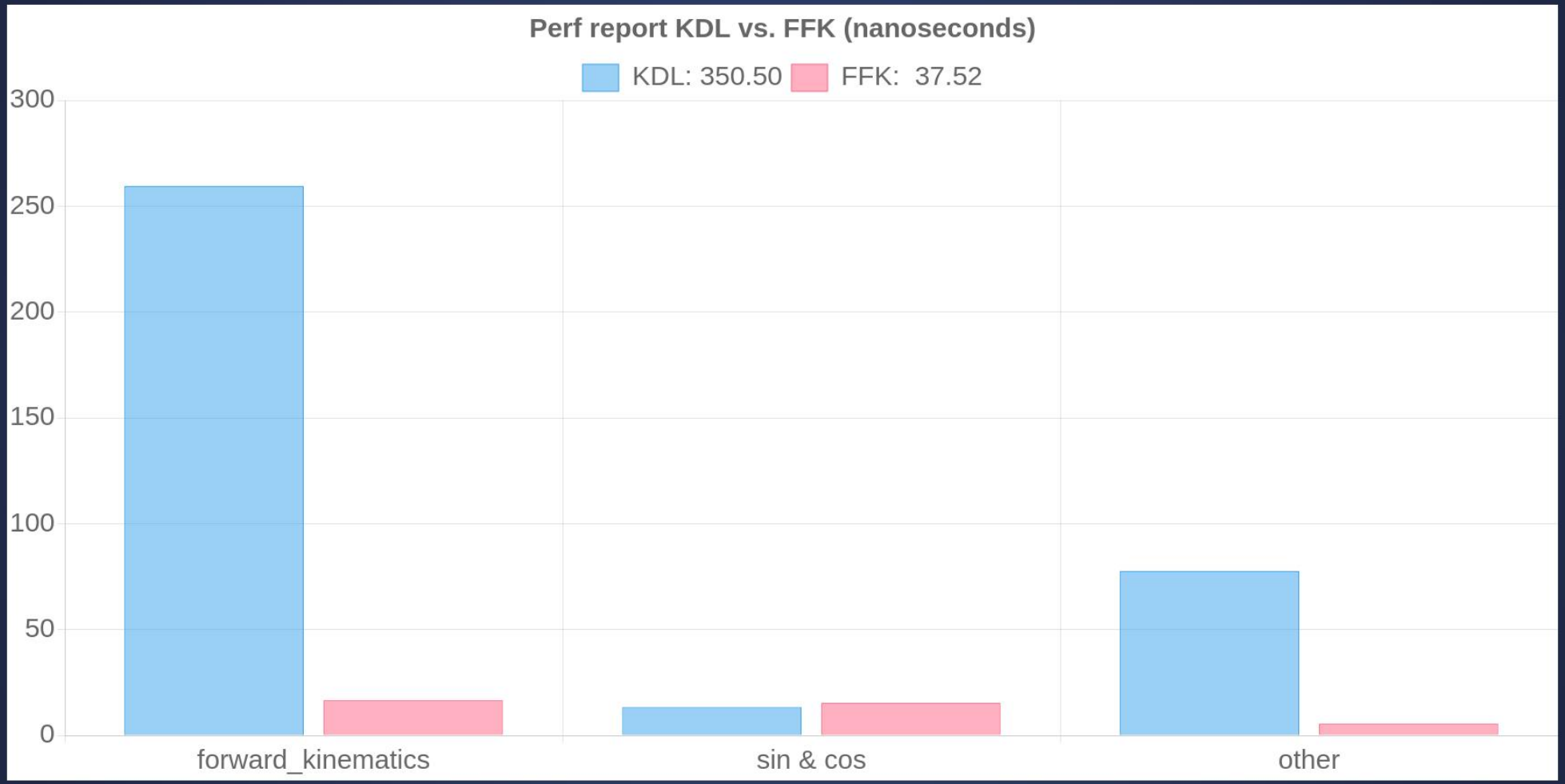
```
705         // dJ3/dq = dJ/ddelta_z * ddelta_z/dq
706
707         // dJ/ddelta_x : 1x3 2*(x_target - x_current)
708         // dJ/ddelta_y : 1x3 2*(y_target - y_current)
709         // dJ/ddelta_z : 1x3 2*(z_target - z_current)
710
711         // ddelta_x/dq: 3 x dofs J_{:3,:} + dofs J_{3:,:} x x_axis
712         // ddelta_y/dq: 3 x dofs J_{:3,:} + dofs J_{3:,:} x y_axis
713         // ddelta_z/dq: 3 x dofs J_{:3,:} + dofs J_{3:,:} x z_axis
714
715
716         // gradient
717         grad *= 0;
718         Eigen::Vector3<float> dJ_ddelta_x = 2 * delta_x;
719         Eigen::Vector3<float> dJ_ddelta_y = 2 * delta_y;
720         Eigen::Vector3<float> dJ_ddelta_z = 2 * delta_z;
721
722         Eigen::Vector<float, 3> joint_pose;
723         Eigen::Vector<float, 3> delta;
724         Eigen::Vector<float, 3> joint_axis;
725         Eigen::Vector<float, 3> jac_linear;
726         Eigen::Vector<float, 3> jac_angular;
727         Eigen::Vector3<float> ddelta_x_dq_ind;
728         Eigen::Vector3<float> ddelta_y_dq_ind;
729         Eigen::Vector3<float> ddelta_z_dq_ind;
730
731
732         {
733             constexpr std::size_t ind = 0;
734
735             joint_pose[0] = input_data[ind * data_size + 2];
736             joint_pose[1] = input_data[ind * data_size + 3];
737             joint_pose[2] = input_data[ind * data_size + 4];
738             delta = ee_pose - joint_pose;
739             joint_axis[0] = input_data[ind * data_size + 7];
740             joint_axis[1] = input_data[ind * data_size + 10];
```

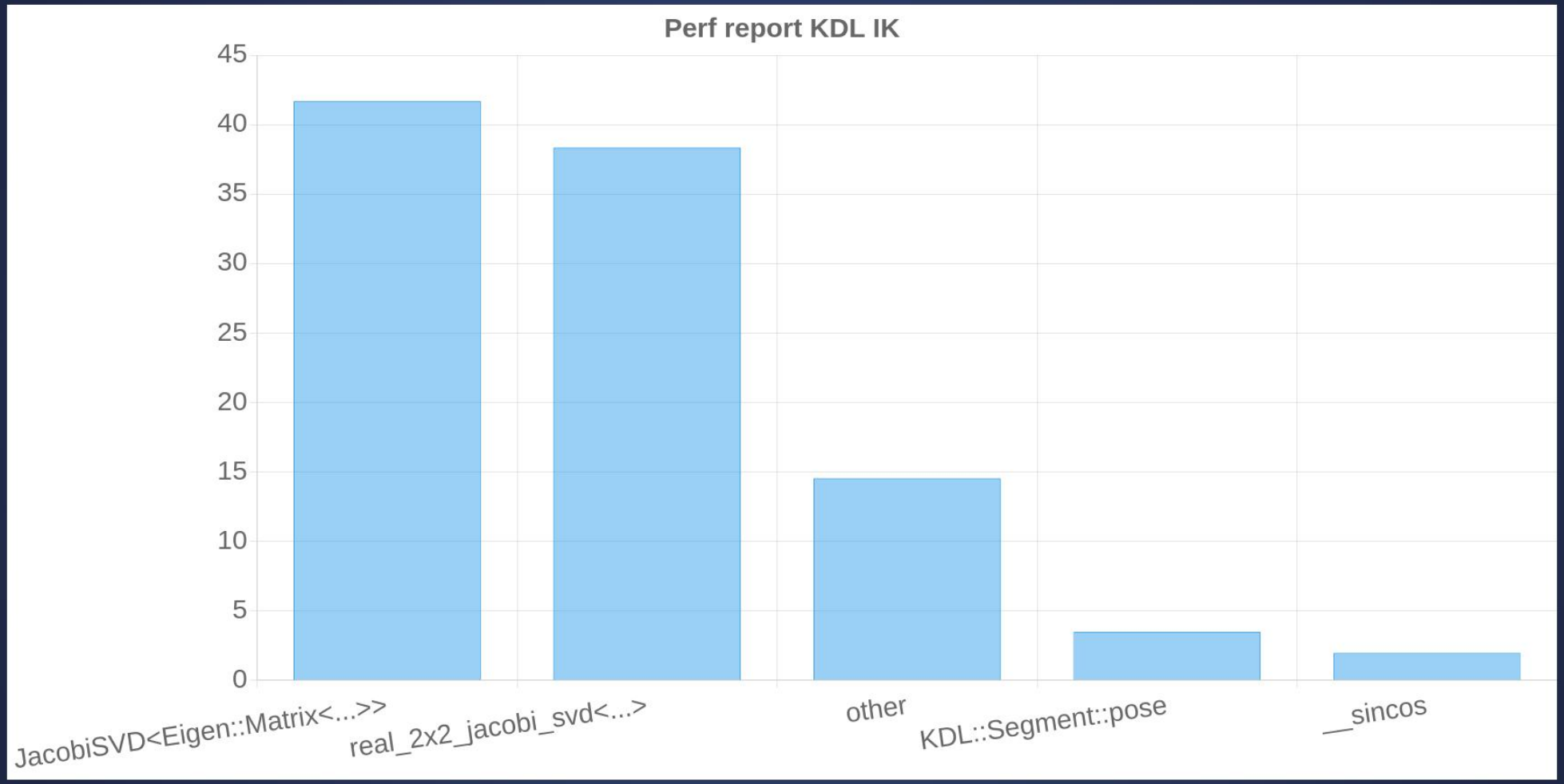
```
        Eigen::Vector3<float> jac_angular;
        Eigen::Vector3<float> ddelta_x_dq_ind;
        Eigen::Vector3<float> ddelta_y_dq_ind;
        Eigen::Vector3<float> ddelta_z_dq_ind;


        {
            constexpr std::size_t ind = 0;

            joint_pose[0] = input_data[ind * data_size + 2];
            joint_pose[1] = input_data[ind * data_size + 3];
            joint_pose[2] = input_data[ind * data_size + 4];
            delta = ee_pose - joint_pose;
            joint_axis[0] = input_data[ind * data_size + 7];
            joint_axis[1] = input_data[ind * data_size + 10];
            joint_axis[2] = input_data[ind * data_size + 13];
            jac_linear = joint_axis.cross(delta);
            jac_angular = joint_axis;

            // x point
            ddelta_x_dq_ind = jac_linear + jac_angular.cross(axis_scale * x_axis);
            grad[ind] -= dJ_ddelta_x.dot(ddelta_x_dq_ind);

            // y point
            ddelta_y_dq_ind = jac_linear + jac_angular.cross(axis_scale * y_axis);
            grad[ind] -= dJ_ddelta_y.dot(ddelta_y_dq_ind);

            // z point
            ddelta_z_dq_ind = jac_linear + jac_angular.cross(axis_scale * z_axis);
            grad[ind] -= dJ_ddelta_z.dot(ddelta_z_dq_ind);


        }

        {
            constexpr std::size_t ind = 1;
```

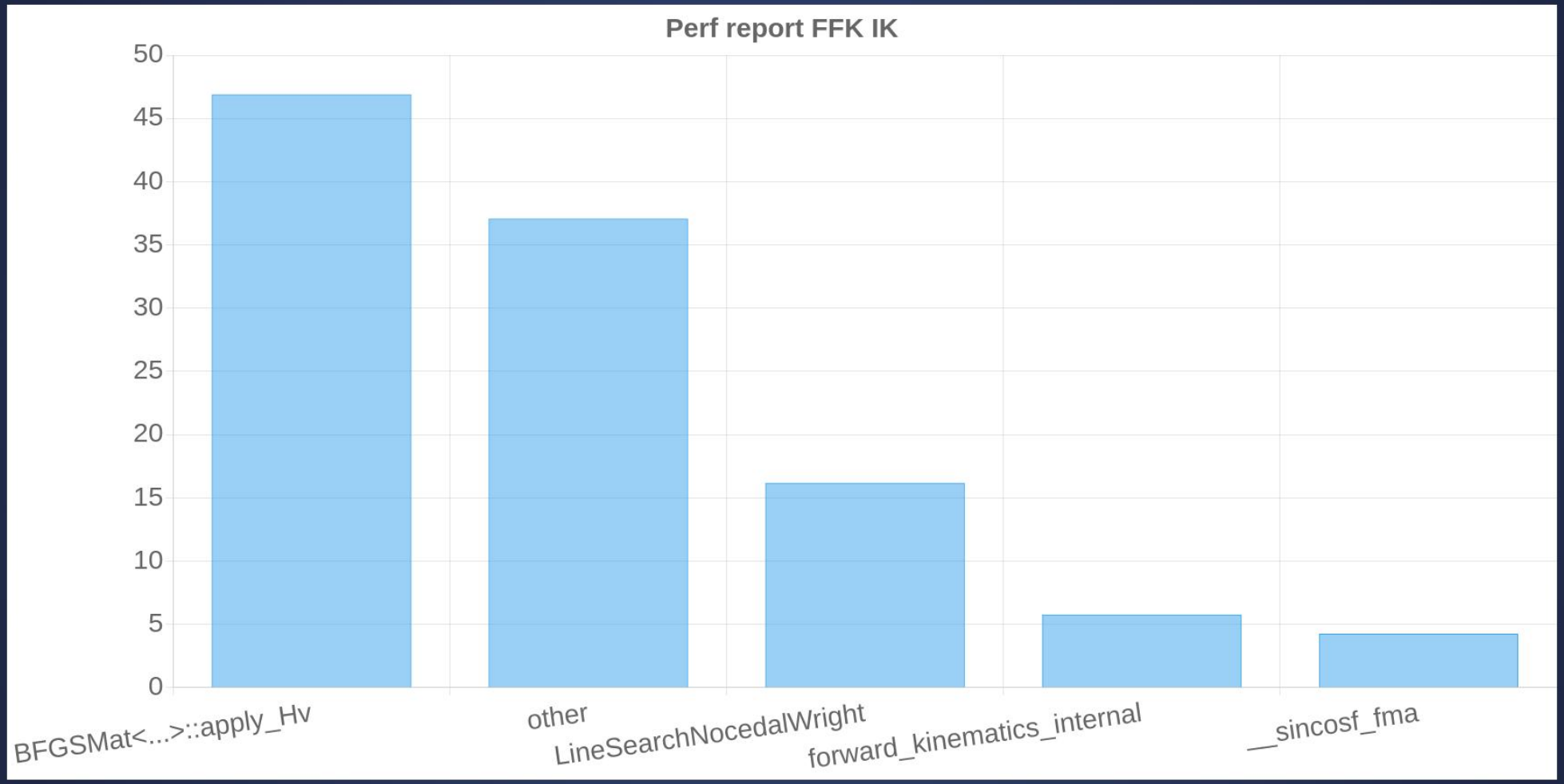
THE RESULTS

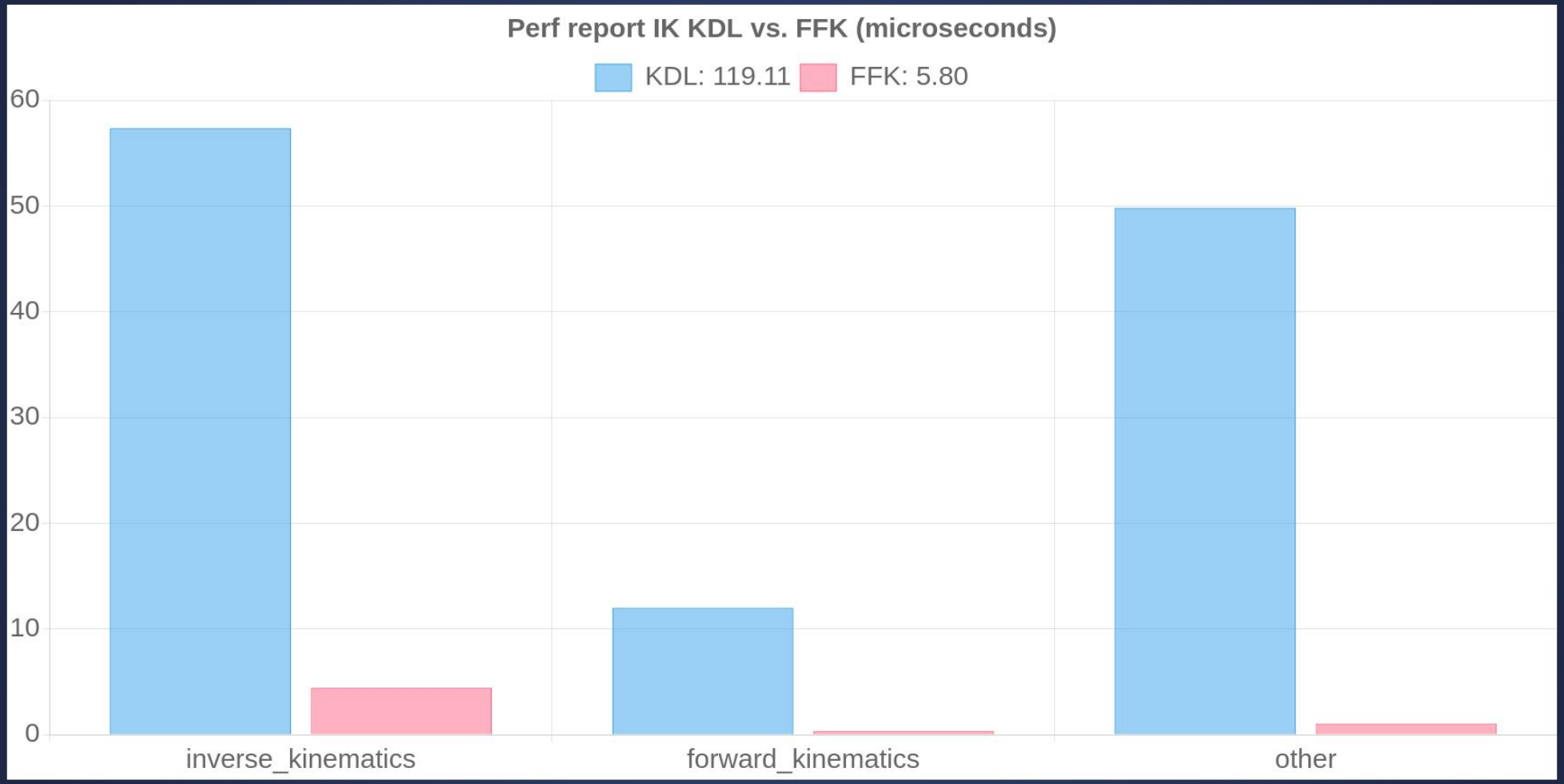

Perf report KDL
35
30
25
20
15
10
5
0
KDL::Segment::pose
KDL::operator*
other
KDL::Joint::pose
KDL::Rotation::Rot2
sin & cos

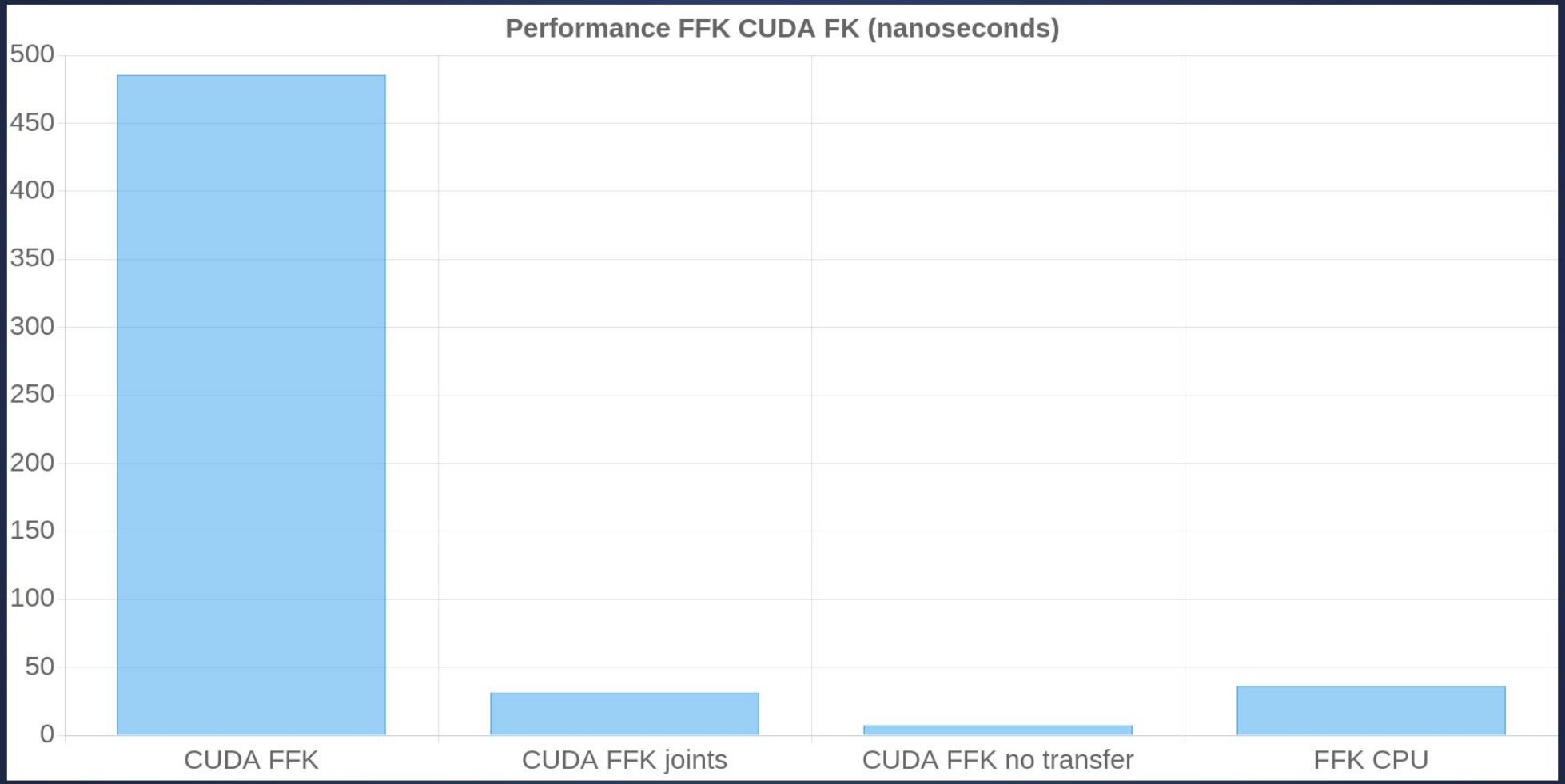

Perf report FFK
45
40
35
30
25
20
15
10
5
0
forward_kinematics_internal
sin & cos
JointData::set_joints
other


Perf report KDL vs. FFK (nanoseconds)
KDL: 350.50
FFK: 37.52
300
250
200
150
100
50
0
forward_kinematics
sin & cos
other


Perf report KDL IK
45
40
35
30
25
20
15
10
5
0
JacobiSVD<Eigen::Matrix<...>>
real_2x2_jacobi_svd<...>
other
KDL::Segment::pose
__sincos


Perf report FFK IK
50
45
40
35
30
25
20
15
10
5
0
BFGSMat<...>::apply_Hv
other
LineSearchNocedalWright
forward_kinematics_internal
__sincosf_fma


Perf report IK KDL vs. FFK (microseconds)
KDL: 119.11
FFK: 5.80
60
50
40
30
20
10
0
inverse_kinematics
forward_kinematics
other


Performance FFK CUDA FK (nanoseconds)
500
450
400
350
300
250
200
150
100
50
0
CUDA FFK
CUDA FFK joints
CUDA FFK no transfer
FFK CPU

Summary: lessons learned

- Leverage compile time optimization when needed in common robotics subroutines
- Leverage code generation to squeeze out as much performance as possible from both the CPU optimizations and known hardware kinematic description
- Try to optimize memory layout and algorithmic improvements

Thank you!

Questions?