MoveIt

Contents

1 – Introduction To Moveit

Moveit ဆိုတာကတော့ Robotic Manipulator တွေရဲ့ Manipulation အတွက်အသုံးချတဲ့ open-source robotics platform တစ်ခုပဲဖြစ်ပါတယ် ။ သူ့ကို Industrial Robots တွေမှာလည်း ကျယ်ကျယ်ပြန့်ပြန့် အသုံးချ ပါတယ် ။ ဒီအောက်က link မှာ MoveIt ကို အသုံးချတဲ့ Robots တွေကို သွားကြည့်လို့ ရ တဲ့အပြင် စိတ်ပါရင် အဲ့ဒီ Robots တွေကို စမ်းဖို့အတွက် Package တွေလည်း အဆင်သင့်ရှိပြီးသားပါ ။

(https://moveit.ros.org/robots/)

Why MoveIt ?

ROS မှာ MoveIt က Motion Planning Framework တစ်ခုဖြစ်ပါတယ် ။ သူ့ Framework ထဲမှာ အောက်က Fig (1.1) မှာပါတဲ့အရာတွေအတွက် အလွယ်တကူအသုံးချလို့ရတဲ့ GUI Tools တွေအပြင် Packages တွေလည်း အများအပြားပါဝင်လို့ MoveIt ကို သုံးရခြင်းဖြစ်ပါတယ် ။

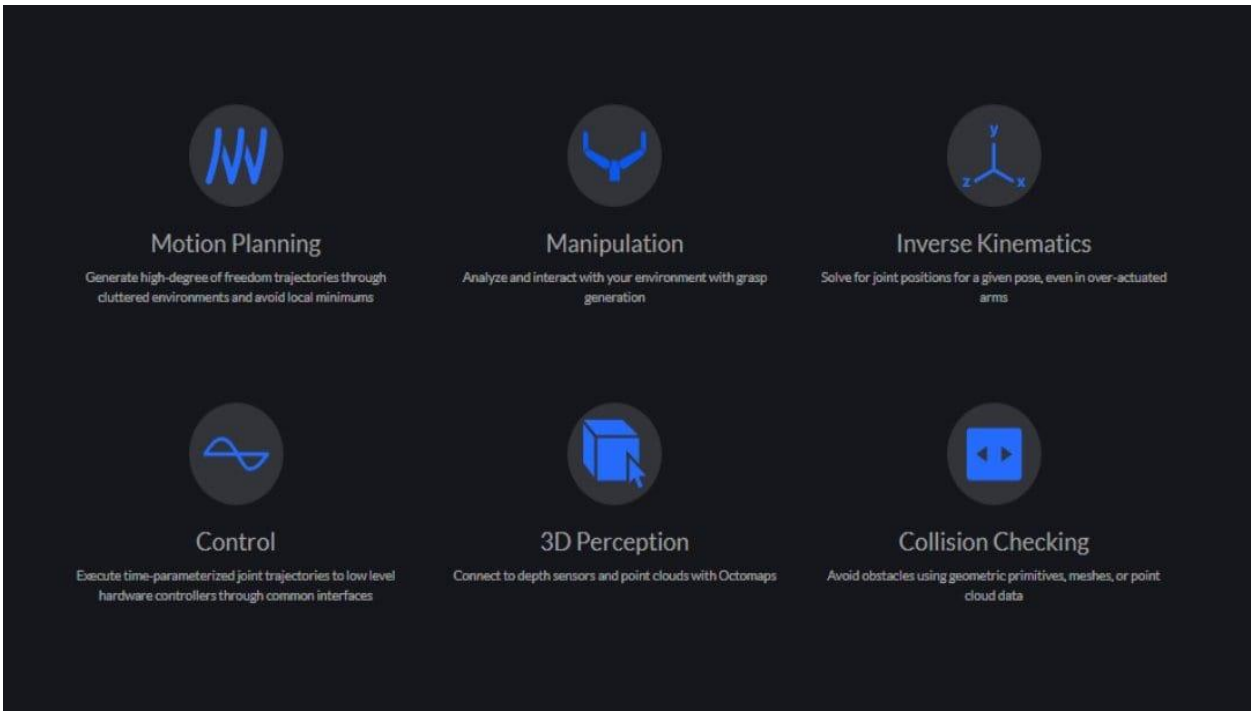


Fig – 1.1

2 – MoveIt Concept

အောက်က Fig (2.1) ကတော့ MoveIt System တစ်ခုလုံးရဲ့ Architecture Diagram ပဲဖြစ်ပါတယ် ။ ကျနော်နောက်ပိုင်းဆက်ရှင်းပြတဲ့ အကြောင်းအရာတွေကို နားလည်တဲ့အခါ ဒီ Diagram ကို သဘောပေါက်သွားမှာပါ ၊ စိတ်မပူပါနဲ့ အခုက အမြည်းကျွေးတာပါ ။

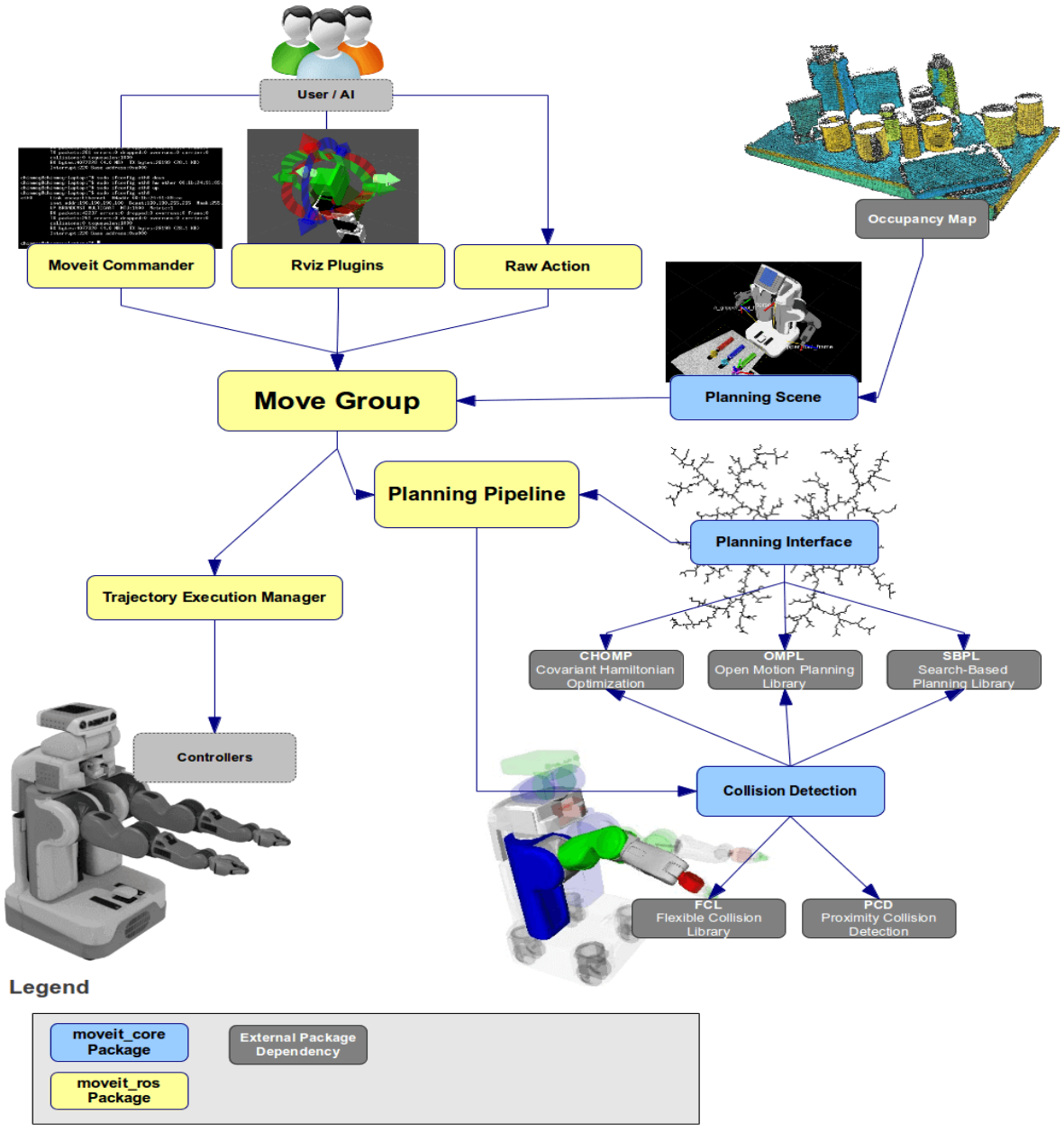


Fig – 2.1

2.a – move_group node

move_group node သည် MoveIt System တစ်ခုလုံးအတွက် Primary Node ကြီးပါ။ဒီ Node ရဲ့ထူးခြားချက်ကတော့ သူက integrator တစ်ခုသာဖြစ်ပါတယ် ၊ Integrator လို့ပြောရခြင်းကတော့ သူသည် ဘယ် Motion Planning Algorithm ကိုမှ run မပေးဘဲ မိမိတို့ရဲ့ Manipulator ကို Motion Planning လုပ်နိုင်ဖို့အတွက် လိုအပ်တဲ့ components တွေရဲ့ function တွေကိုသာ သူ့ Node ထဲမှာ Plugins အဖြစ် connect လုပ်ပေးခြင်းပဲဖြစ်ပါတယ် ။ သူ့ချည်းသက်သက်ကတော့ integrator သာဖြစ်လို့ ဘာမှ မလုပ်ပေးနိုင်ပါဘူး ။ component ဆိုတာကတော့ Robot ရဲ့ current state(pose), joint states တွေ ၊ 3D sensors တွေကရတဲ့ sensor data တွေ ၊ ပြီးတော့ ROS Parameter Server ကရတဲ့ Robot Model ရဲ့ Information တွေနဲ့ user`s inputs တွေပဲဖြစ်ပါတယ်။အောက်က Fig (2.2) မှာကြည့်ရင် ရှင်းသွားမယ် ထင်ပါတယ် ။

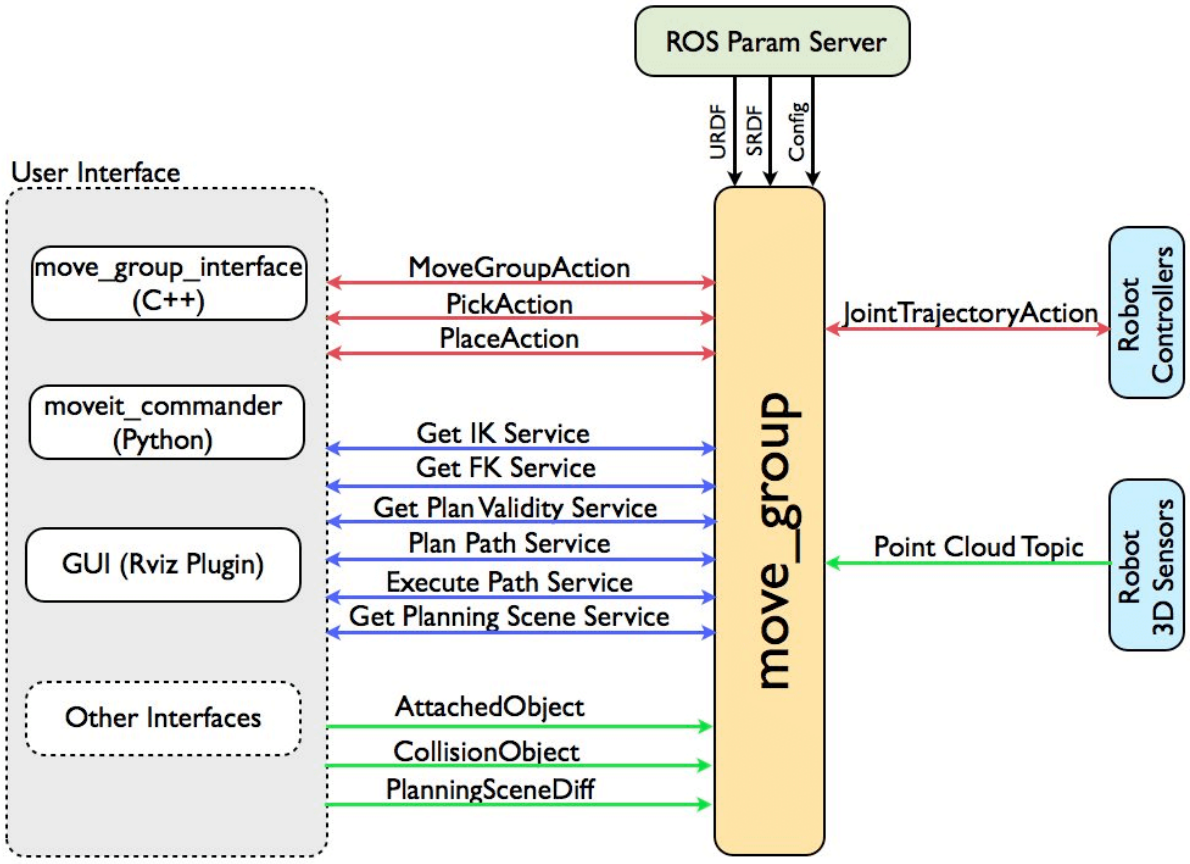


Fig – 2.2

မရှင်းသေးလည်း စိတ်မပူပါနဲ့ ၊ အောက်မှာ အသေးစိတ်ကို သေချာလေး ရှင်းမှာပါ ။

2.a.(i) – Userface Interface

ဒီခေါင်းစဉ်မှာ ရှင်းမှာကတော့ အောက်က Fig (2.3) မှာပါတဲ့ အနီရောင်လေးထောင့်ကွက်ထဲက အပိုင်းပါ ။

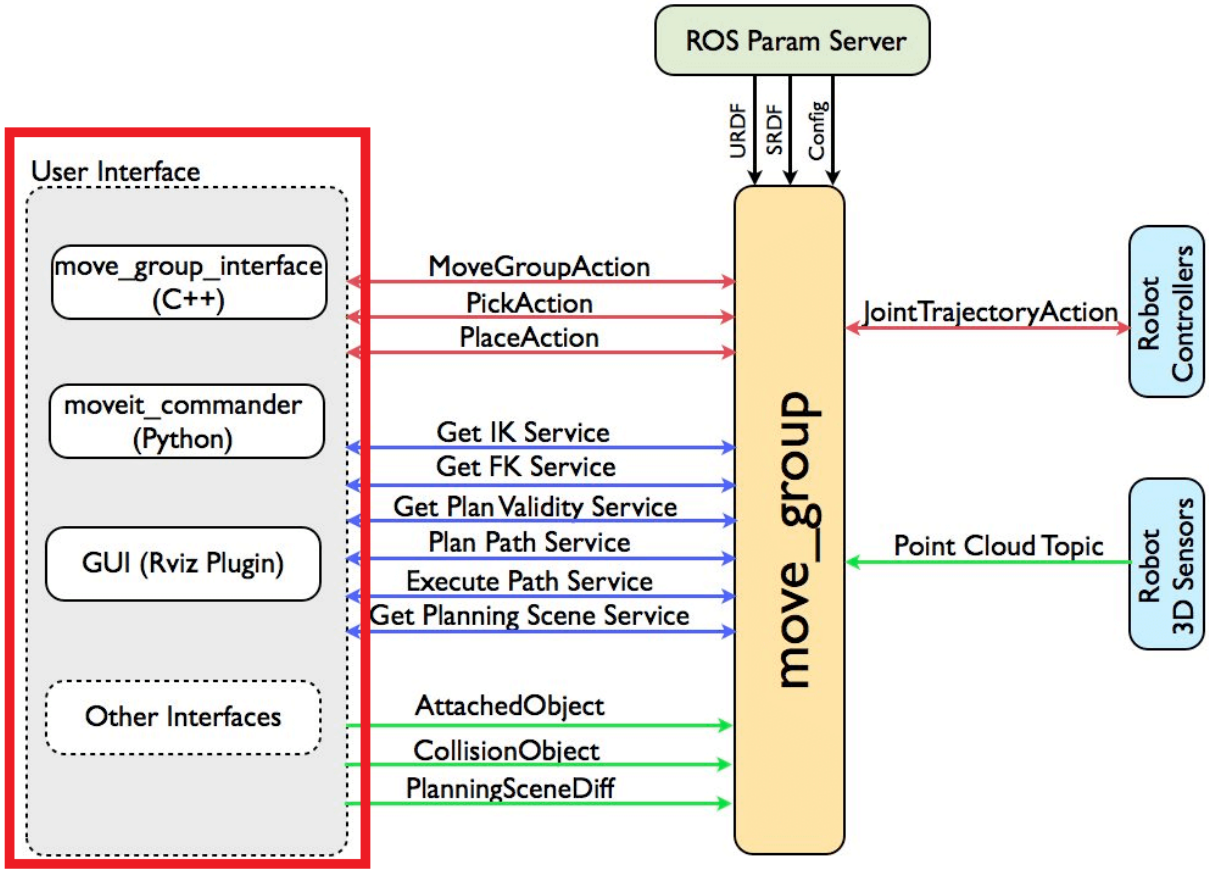


Fig – 2.3

User Interface ဆိုတာကတော့ move_group node ကို Program ရေးခြင်း သို့ GUI တစ်ခုခုသုံးပြီး မိမိတို့ လိုချင်တဲ့ Action ကို command ပေးတဲ့ အပိုင်းပဲဖြစ်ပါတယ် ။ ဥပမာအနေနဲ့ပြောရမယ်ဆို Pick and Place လုပ်တဲ့ action ပေါ့ ။ command ပေးလို့ရတဲ့ interface သုံးခုရှိပါတယ် ။ အောက်မှာကြည့်ပါ

- **C++ Interface**
- **Python Interface**
- **Rviz Motion Planning Plugin (GUI)**

2.a.(ii) – Configuration

ဒီခေါင်းစဉ်မှာ ရှင်းမှာကတော့ အောက်က Fig (2.4) မှာပါတဲ့ အနီရောင်လေးထောင့်ကွက်ထဲက အပိုင်းပါ ။

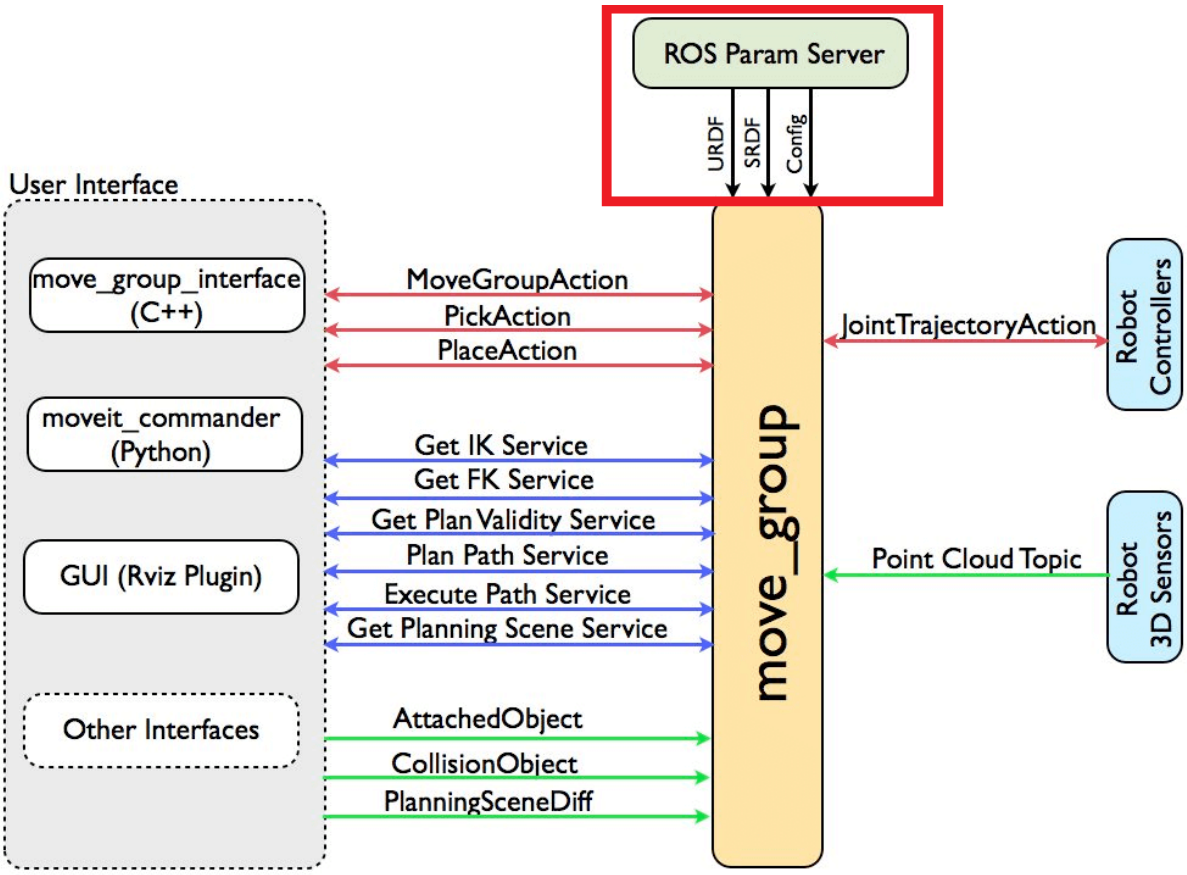


Fig – 2.4

ဒီအပိုင်းမှာတော့ move_group node သည် Parameter Server ပေါ်မှာရှိတဲ့ အောက်ပါ Information တွေကို ရယူအသုံးပြုပါတယ် ။

- **URDF (Unified Robot Description Format)**
- **SRDF (Semantic Robot Description Format)**
- **Moveit Configurations**

စိတ်မပူပါနဲ့ အပေါ်က ဒီသုံးခုလုံးကို အောက်မှာ အသေးစိတ် ဆက်ရှင်းမှာပါ ။

❖ URDF

move_group node သည် ROS Parameter Server ပေါ်မှာရှိတဲ့ "robot_description" ဆိုတဲ့ Parameter လေးကို ရှာပြီး Robot Model အတွက် URDF ကို ရယူပါတယ် ။ MoveIt ထိ ရောက်လာပြီးမှတော့ URDF အကြောင်းကိုလည်း လေ့လာပြီးပြီလို့ ထင်ပါတယ် ၊ အကယ်၍ မလေ့လာရသေးရင် URDF ကို ကျနော် မြန်မာလိုရေးထားတဲ့ ဒီ Link မှာ PDF ကို Download ပြီး လေ့လာနိုင်ပါတယ် ။
(https://drive.google.com/file/d/1tkPKAGPv2nqGSMGMmqdAxxnZmiLoJ9kD/view)

❖ SRDF

move_group node သည် ROS Parameter Server ပေါ်မှာရှိတဲ့ "robot_description_semantic" ဆိုတဲ့ Parameter လေးကို ရှာပြီး Robot Model အတွက် SRDF ကို ရယူပါတယ် ။ SRDF ကို Moveit ရဲ့ အားသာချက် Tool တစ်ခုဖြစ်တဲ့ Moveit Setup Assistant ကနေပြီး Generate လုပ်ပေးပါလိမ့်မယ် ။

What is SRDF ?

SRDF သည် URDF ထဲမှာ မပါတဲ့ Information တွေကို သတ်မှတ်ပေးပါတယ် ၊ ရှင်းအောင်ပြောရရင် Robot Model ကိုဖော်ပြမယ့် URDF ကို ပိုပြည့်စုံသွားစေတယ်ပေါ့ဗျာ ။ ဘယ်လို Information တွေလဲဆိုတာ အောက်မှာကြည့်ပါ ။

- **Self-Collision –** Self-Collisions ဆိုတာကတော့ မိမိတို့ Robot ရဲ့ paired link အချင်းချင်း collision ဖြစ်နေလားဆိုတာ check လုပ်တာပါ ၊ ဥပမာ link 1 နဲ့ Link 6 ကတော့ self collision မဖြစ်ဖူးဆိုရင် သူတို့ကို Never in collision လို့သတ်မှတ် လိုက်မယ် ၊ ပြီးတော့ link 2 နဲ့ link 3 က collision ဖြစ်နိုင်တဲ့ link တွေဆိုရင် သူတို့ကို adjacent link collision လို့သတ်မှတ်လိုက်မယ်။ ဒီလို self collision checking လုပ်ခြင်းအားဖြင့် သတ်မှတ်ခြင်းအားဖြင့် motion planning လုပ်တဲ့အချိန်ကိုလျှော့ချနိုင်တာပေါ့ ။ ထိုအလုပ်ကို Moveit ရဲ့ Default ပါတဲ့ Self-Collision Matrix Generator က လုပ်ပေးပါတယ် (အဲ့ Generator ကဘာလဲသိချင်ရင် ကိုယ့်ဘာသာ ရှာကြည့်ပါ) ။

Self Collision Checking ကို MoveIt Setup Assistant ထဲ Generate လုပ်တဲ့အခါမှာ Sampling Density ဆိုတဲ့ဟာလေး တွေ့ပါလိမ့်မယ် ၊ Default value က 10000 ပါ ။ Sampling Density Value ဆိုတာကတော့ self collision checking အတွက် link တွေကို random position ဘယ်နှကြိမ်ထိ sample ယူပြီး train မှာလည်း check မှာလည်း ဆိုတာကို သတ်မှတ်တာပါ ။ sampling density value များလေလေ computation time ပိုကြာလေလေဖြစ်ပါတယ် ။

- **Virtual Joints -** Virtual Joint သည် မိမိတို့ Manipulator ရဲ့ first link ကို World Coordinate System မှာ Attach လုပ်ပေးတဲ့ Fixed Joint ပဲဖြစ်ပါတယ် ။

- **Passive Joints -** Passive Joint ဆိုတာကတော့ မိမိတို့ Manipulator မှာရှိတဲ့ တချို့ Joint တွေကို unactuate ဖြစ်စေချင်တဲ့အခါ သုံးပါတယ် ၊ Manipulator မှာဆို Joint အားလုံးက actuate ဖြစ်တာမို့ အများအားဖြင့် သိပ်သတ်မှတ်လေ့တော့ မရှိပါဘူး ၊ မြင်အောင် ဥပမာပေးရရင် Differential Wheeled Mobile Robot တွေမှာပါတဲ့ Caster Wheel ရဲ့ Joint က Passive Joint ပေါ့ ၊ Motor Wheel ရဲ့ Joint တွေကတော့ Actuate ဖြစ်တဲ့အတွက် Passive Joint မဟုတ်ဖူးပေါ့ ၊ ဒီလောက်ဆို မြင်မယ်ထင်ပါတယ် ။

- **Planning Groups -** Planning Group - ဆိုတာကတော့ မိမိတို့ Manipulator မှာ Same Actuation လုပ်ဆောင်စေချင်တဲ့ Link သို့ Joint တွေကို Group သတ်မှတ်ပေးခြင်းပါ ။ ရှင်းအောင်ထပ်ပြောရမယ်ဆို မိမိတို့ Manipulator မှာဆို End-Effector မှာပါတဲ့ Link တွေ Joint တွေကို Hand Group လို့ သတ်မှတ်ပေး ပြီး Manipulator ရဲ့ ကျန်တဲ့အစိတ်အပိုင်းတွေရဲ့ Link တွေ Joint တွေကို Arm Group လို့ သတ်မှတ်ပေးခြင်းပါ ။ ဒီလိုသတ်မှတ်မှသာလျှင် မိမိတို့ Manipulator ရဲ့ အစိတ်အပိုင်းတွေကို စိတ်ကြိုက်ထိန်းချုပ်လို့ရမှာဖြစ်ပါတယ် ။ နောက်ထပ် လုပ်လို့ရတာလေးကတော့ Sub-Group လေးတွေသတ်မှတ်လို့ရခြင်းပါ ။ ဥပမာပေးရရင် ကျနော်တို့ Arm Group မှာ Joint တွေ Link တွေ အများကြီးရှိမယ် ၊ အဲ့ထဲကမှ Joint ဘယ်နှခု Link ဘယ်နှခုကို New Group (Sub-Group) လေး သတ်သတ်ထပ်ဖွဲ့ခြင်းကို ဆိုလိုတာပါ ။ မိခင် Arm Group ရဲ့ အခွဲလေးပေါ့ ။

- **End-Effectors -** End-Effectors ဆိုတာကတော့ မိမိတို့ Manipulator မှာရှိတဲ့ End-Effector ရဲ့ link တွေ Joint တွေကို Special Group အဖြစ် သတ်မှတ်ပေးခြင်းပါ။ ခုနက ကျနော် Planning Group မှာပြောခဲ့တဲ့ Hand Group ကိုပဲ ဆိုလိုတာပါ ။ ဒီမှာ ပြောစရာရှိတာက တူနေရင် ဘာလို့ ထပ်သတ်မှတ်လဲပေါ့ ။ ကျနော်မြင်တာကိုပြောရရင် End-Effector ကို သူ့အလုပ်ဖြစ်တဲ့ Object Pick and Place ကို ပိုပြီး အမှားအယွင်းမရှိ သူ့အလုပ် သူလုပ်နိုင်အောင်သတ်မှတ်တယ်လို့ ထင်ပါတယ် ၊ End-Effector တွေတစ်ခုထက်ပိုတဲ့ Manipulator တွေလည်း ရှိတယ်ဆိုတော့ ဒါမျိုးသတ်မှတ်ပေးမှ specific task ကို လုပ်ဆောင်လို့ရမယ်လို့ ထင်ပါတယ် ။

- **Robot Poses -** Robot Poses ဆိုတာကတော့ မိမိတို့ Robot ရဲ့ Initial Pose ကို သတ်မှတ်ပေးခြင်းပါ ။ ရည်ရွယ်ချက်ကတော့ Real World မှာ Manipulator စစချင်းထားမယ့် Pose နဲ့ ကျနော်တို့ ROS MoveIt (In Rviz) ထဲက Pose နဲ့ အနီးစပ်ဆုံးတူအောင် သတ်မှတ်တာပါ ။ထိုသို့ သတ်မှတ်လိုက်တဲ့အခါ မိမိတို့ ROS ဘက်က command ပေးတာကို Real World က Manipulator ကောင်းကောင်း လုပ်ဆောင်နိုင်မှာဖြစ်ပါတယ် ။

အထက်ပါပြောခဲ့တဲ့ SRDF Information တွေအကုန်လုံးကို Moveit Setup Assistant ဆိုတဲ့ Tool က Generate လုပ်ပေးပါလိမ့်မယ် ။ စိတ်မပူပါနဲ့ ။ MoveIt Setup Assistant ဆိုတာ ကြိမြင်ထားချင်ရင် အောက်က Fig (2.5) မှာကြည့်ပါ ။

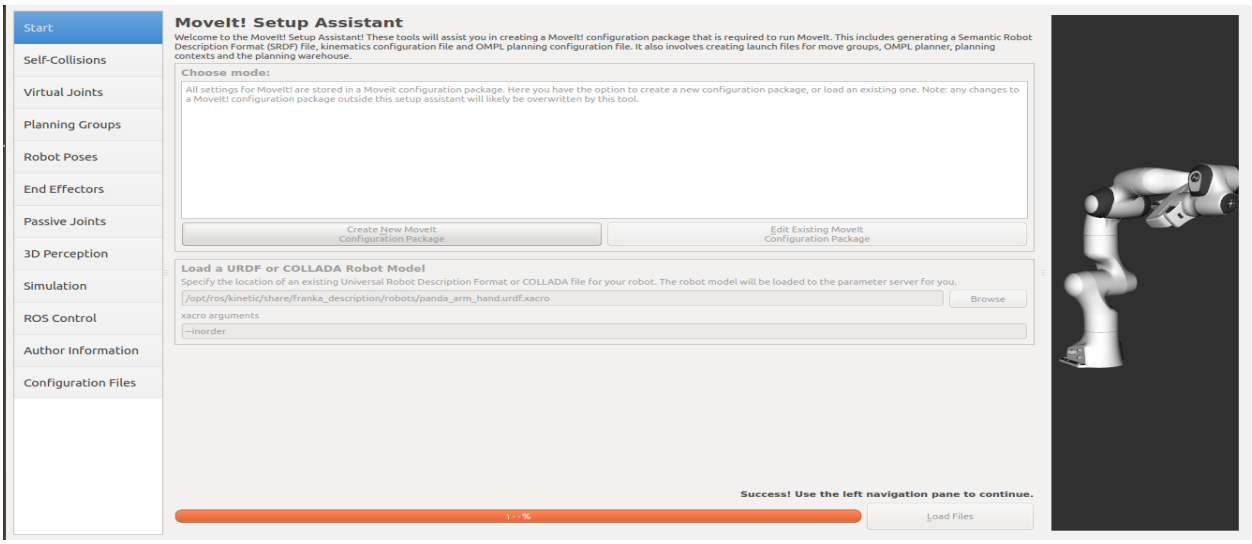


Fig – 2.5

❖ Moveit Configurations

move_group node သည် ROS Parameter Server ပေါ်မှ အပေါ်က URDF,SRDF အပြင် တခြားသော configuration file တွေဖြစ်တဲ့ joint limits , kinematic plugins , motion planning plugins တွေစသဖြင့် YAML Files တွေကိုလည်း ရယူပါတယ် ။ Moveit Setup Assistant ဆိုတဲ့ Tool က Generate လုပ်ပေးပါလိမ့်မယ် ။ စိတ်မပူပါနဲ့ ။

2.a.(iii) – Robot Interface

ဒီခေါင်းစဉ်မှာ ရှင်းမှာကတော့ အောက်က Fig (2.6) မှာပါတဲ့ အနီရောင်လေးထောင့်ကွက်ထဲက အပိုင်းပါ ။

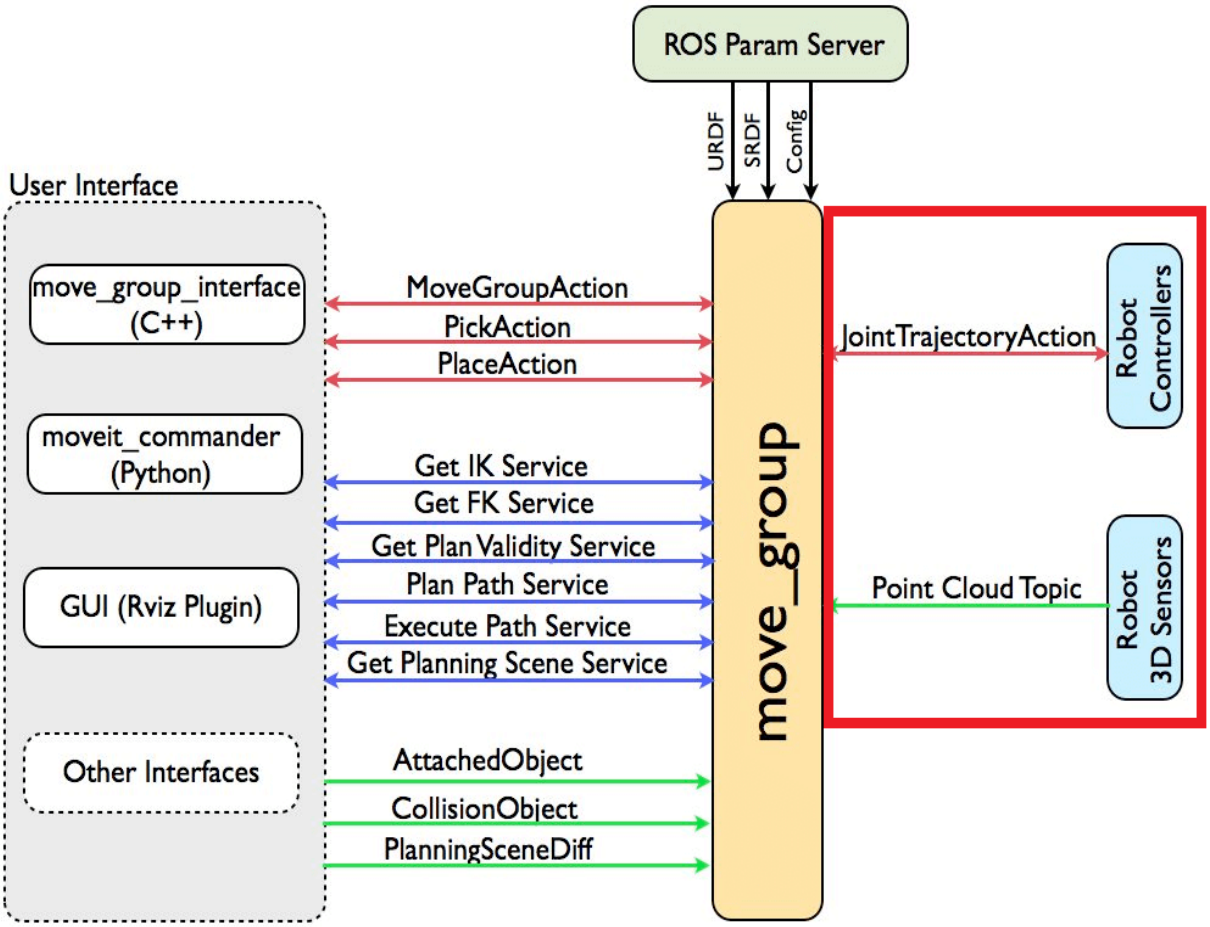


Fig – 2.6

ဒီအပိုင်းမှာတော့ move_group node သည် ROS Topics တွေသုံးပြီး အောက်ပါ အချက် သုံးချက်ကို Sensor တွေ သက်ဆိုင်ရာ Node တွေဆီကနေ ရယူပြီး ROS ရဲ့ Action Infertace တစ်ခုဖြစ်တဲ့ "FollowJointTrajectoryAction" ကိုသုံးပြီး မိမိတို့ Robot Controllers တွေဆီကို Talk လုပ်ပါတယ် ။ Action Inferface ဖြစ်တဲ့အတွက် Server နဲ့ Client ရှိမယ်ဆိုတာ သိပြီးသားဖြစ်မှာပါ ။ Server ကတော့ Robot Controller မှာဖြစ်ပြီး client ကတော့ move_group node ကနေ initiate လုပ်ရမှာပါ။ အဲ့သုံးချက်ကို အောက်မှာ သေချာလေးပြောပြပါမယ် ၊ ဆက်ကြည့်ပါ ။

- **Point Clouds and Other Sensor Datas -** ဒီအချက်ကတော့ ရှင်းပါတယ် ၊ move_group node သည် မိမိတို့ Robot 3D Sensors တွေကနေတဆင့် Sensor Data တွေကို ရယူပါတယ် ။

- **Joint State Information -** ဒီအချက်ကတော့ move_group node သည် joint_state-publisher node ကနေ publish လုပ်ပေးတဲ့ "/joint_states" ကို listen လုပ်ပြီး robot ရဲ့joint positions (joint states) ကို ရယူပါတယ် ။ ရယူတဲ့နေရာမှာ move_group node က joint_state_publisher ကို သူ့ဘာသာ Setup မလုပ်တဲ့အတွက်ကြောင့် User တွေက မိမိတို့ Package ရဲ့ launch file ထဲမှာ joint_state_publisher node ကို ရေးပေးဖို့လိုပါတယ် ။

- **Transform Information -** ဒီအချက်ကတော့ move_group node သည် TF Library ကိုသုံးပြီး transform information တွေကိုရယူပါတယ် ၊ ဒီထိရောက်လာပြီးမှတော့ tf ကို သိမယ်လို့ ထင်ပါတယ်။ သတိထားရမှာက move_group node က TF ကို listen ပဲ လုပ်ပါတယ် ၊ publish မလုပ်ပါဘူး ။ TF ကို Publisher လုပ်ဖို့အတွက် User တွေက မိမိတို့ Package ရဲ့ launch file ထဲမှာ robot_state_publisher node ကို ရေးပေးဖို့လိုပါတယ် ။

move_group node ကနေ အပေါ်က Information တွေရယူပြီး Robot Controller တွေဆီ Talk လုပ်တယ်လို့ ပြောခဲ့ပါတယ် ။ ထို့ကြောင့် Robot Controller ဆိုတာ ဘာလဲသိဖို့လိုလာပါတယ် ။ Robot Controller တွေအကြောင်းကို wiki.ros.org နဲ့ gazebosim.org တို့မှာလည်း လေ့လာနိုင်သလို ကျနော် မြန်မာလိုရေးထားတဲ့ ဒီအောက်က link မှာ PDF Download ပြီးလေ့လာနိုင်ပါတယ် ။

(https://drive.google.com/file/d/1c7e1u32h8l9yCzXkT1SSKn3mj17vkR-v/view?fbclid=IwAR2RJU9KGe5wSxTZj9eqIRp_6Q5ND87Y7q9QdXwPBOWGmSSNjo4GWf1C4uE)

2.b – Motion Planning

Introduction မှာ MoveIt ဆိုတာ Motion Planning Framework တစ်ခုလို့ ကျနော် မိတ်ဆက်ခဲ့ပါတယ် ။ MoveIt ရဲ့ နောက်ဆုံး Goal ကိုက အကောင်းဆုံး Trajectory တစ်ခုကို Robot Controller တွေဆီ ပို့ပေးဖို့ပဲဆိုတာ သဘောပေါက်ဖို့လိုပါတယ် ။ သတိပြုရမှာ trajectory နဲ့ path နဲ့ မတူပါဘူး ။ ကျနော်တို့ ခုပြောနေတဲ့ trajectory ဆိုတာ path+time ပါ ။

2.b.(i) – What is Motion Planning ?

Motion Planning သည် မိမိတို့ Manipulator သွားမယ့် Start Pose ကနေ Goal Pose တစ်လျှောက် Joint Limit တွေကို မကျော်တဲ့ ၊ World ထဲက မည်သည့် Obstacle နဲ့မှမထိတဲ့ ၊ သူ့ရဲ့ Link အချင်းချင်းလည်း collide မဖြစ်တဲ့ အသင့်တော်ဆုံး အကောင်းဆုံး လမ်းကြောင်းတစ်ခုကို ရှာတဲ့ Technique တစ်ခုပါ ။ ကျနော်တို့မှာ Motion Planning အတွက်လိုအပ်တဲ့ Information တွေဖြစ်တဲ့ Robot ရဲ့ initial pose,goal pose, geometrical description နဲ့ world ရဲ့ geometrical description တော့ရှိဖို့လိုပါတယ် ။

2.b.(ii) – Motion Planning Plugin

MoveIt က Motion Planning Algorithms တွေကို သုံးဖို့ သက်ဆိုင်ရာ Motion Planner တွေနဲ့တူတူ အလုပ်လုပ်ပါတယ် ။ ရှင်းအောင်ထပ်ပြောရရင် move_group node က ROS Actions or ROS Services တွေကတဆင့် Plugin Interface တစ်ခုသုံးပြီး Motion Planner တွေဆီကို Talk လုပ်ပေးခြင်းဖြစ်ပါတယ် ။ စိတ်မပူပါနဲ့ ။ Plugin Interface ကို **Moveit Setup Assistant** ကနေပြီး Generate လုပ်ပေးပါလိမ့်မယ် ။ move_group node အတွက် default motion planner တွေကိုတော့ OMPL(Open Motion Planning Library) ကနေ ရယူပါတယ် ။ဆိုတော့ OMPL မှာရှိတဲ့ planner တွေကို Default Motion Planner အဖြစ်သုံးတယ်ပေါ့ ။ (http://ompl.kavrakilab.org/) ဒီ link က OMPL ရဲ့ website ပါ ။

MoveIt ကို ကိုယ်က သေချာနားမလည်သေးဖူးဆို ကျနော်အောက်မှာဆက်ရှင်းမယ် What is OMPL ဆိုတာလေးကို ကျော်ပြီးဖတ်ပေးပါ ၊နောက်သေချာနားလည်မှ ပြန်လာဖတ်ရင်လည်း ရပါတယ် ၊ နောက်မို့ဆို ပို ရှုပ်သွားမှာစိုးလို့ပါ ၊ moveit official documentation ကနေ အဓိက reference ယူပြီး Concept ကို ရှင်းတာဆိုတော့လေ ။

What is OMPL ?

OMPL မှာ Sampling Based Motion Planning Algorithms တွေ အများအပြား ပါဝင်ပါတယ် ။ Sampling Based Algorithms ဆိုတာကို အကြမ်းဖျဉ်းပုံဖော်ကြည့်ချင်ရင်တော့ ဒီ link (https://www.youtube.com/watch?v=EtH5I5xlyU8) မှာသွားကြည့်လို့ပဲ ညွှန်းချင်ပါတယ် ၊ ပိုပြည့်စုံမှာပါ ။ OMPL က သူ့ချည်းပဲဆို Motion Planning အတွက် collision checking, visualization နဲ့ ပက်သက်တဲ့ မည်သည့် code မှမပါပါဘူး ၊ ဆိုလိုတာက OMPL မှာရှိတဲ့ planner တွေက abstract ဖြစ်တယ်ပေါ့ ၊ planner တွေသာ motion planning လုပ်နိုင်ဖို့ path generate လုပ်နိုင်ဖို့အတွက်ပဲ ဖြစ်ပြီး has no concept of robot လို့ပြောချင်တာပါ ။

ထို့ကြောင့် သူ့ကို တခြားသော robot software platform တွေနဲ့ integrate လုပ်ပြီးမှသာ ကောင်းကောင်း အသုံးပြုလို့ရတာဖြစ်ပါတယ် ။ဥပမာ moveit လိုဟာမျိုးပေါ့ ။ ရှင်းအောင်ထပ်ပြော ရရင်ဗျာ Planner တွေက Path ပဲ Generate လုပ်ပေးတာပါ ။ Moveit Setup Assistant သုံးပြီး Generate လုပ်ပေးရာမှ ရရှိလာတဲ့ joint_limits.yaml ဆိုတဲ့ file ထဲမှာရှိတဲ့ value တွေကိုသုံးပြီး Path ကို Time Parameterization လုပ်ပြီး Trajectory ကို Generate လုပ်ပေးမှာပါ ။

ဒီတစ်ချက်လေး မှတ်ထားရင်ရပါပြီ ။ Planner တွေဆီက ထုတ်ပေးလိုက်တဲ့ Path ကို moveit မှာရှိတဲ့ information တွေနဲ့ တွဲသုံးမှသာလျှင် ကျနော်တို့လိုချင်တဲ့ သတ်မှတ် ထားတဲ့ joint limit အတွင်း collision မဖြစ်ဘဲ မိမိတို့လိုချင်တဲ့ Goal Pose အထိ plan ကောင်းကောင်း လုပ်နိုင်တဲ့ trajectory တစ်ခုကို ရရှိမှာ ဖြစ်ပါတယ် ။ Plan လုပ်လိုက်တဲ့ trajectory ကိုလည်း visualize လုပ်နိုင်မှာဖြစ်ပါတယ်။ဒီ link မှာ robot software platform တွေကိုသွားကြည့်လို့ရပါတယ် ။

(http://ompl.kavrakilab.org/integration.html)

OMPL မှာရှိတဲ့ Planner တွေကိုတော့ category နှစ်ခု ခွဲထားတယ်ဗျ ။ အောက်မှာဆက်ကြည့်ပါ ။

- **Geometric Planners**
- **Control-Based Planners**

Moveit နဲ့ Integrate လုပ်တဲ့အခါ move_group ထဲမှာရှိနေတဲ့ information နဲ့ ကိုက်ညီတဲ့ Planner ကို OMPL က သူ့ထဲရွေးချယ်ပြီး အသုံးပြုပေးမှာဖြစ်ပါတယ် ၊ Planner တွေများလွန်းလို့ ကိုယ့်ဘာသာပဲ ဒီ Link မှာ စိတ်ပါရင် လေ့လာနိုင်ပါတယ်ခမျာ ။ ကျနော် OMPL နဲ့ပက်သက်ပြီးသိတာက လောလောဆယ် ဒီလောက်ပါပဲ ။ (http://ompl.kavrakilab.org/planners.html)

2.b.(iii) – Motion Plan Request

အိုကေ ကျနော်တို့ Motion Planning စလုပ်ကြမယ်ဆို အရင်ဆုံးလိုအပ်မှာက Motion Planner တွေဆီကို ကျနော်တို့လိုချင်တဲ့ Planning Requirement တွေသတ်မှတ်ထားတဲ့ Motion Planning Request ကို ပို့ပေးဖို့လိုမှာ ဖြစ်ပါတယ် ။ Planning Requirement ဆိုတာကတော့ ဥပမာပေးရရင် new goal pose သတ်မှတ်တာတို့ joint location ကို ပြောင်းလဲချင်တဲ့အခါတို့ပေါ့ ။

Motion Planner တွေ check လုပ်ဖို့အတွက် kinematic contraints တွေ သတ်မှတ်ပေးလို့ရပါတယ် ၊ Moveit မှာ inbuilt constraints တွေရှိပြီးသားပါ ၊ အောက်မှာဆက်ကြည့်ပါ ။

- **Position Constraints –** ဒီ contraint ကတော့ link ရဲ့ position ကို အဲ့ link ရှိရမယ့် region အတွင်းမှာပဲရှိနေအောင် ကန့်သတ်ထားတာပါ ။

- **Orientation Constraints – –** ဒီ contraint ကတော့ joint ရဲ့ orientation ကို အဲ့ joint ရှိရမယ့် roll, pitch,yaw angles အတွင်းမှာပဲ ရှိနေအောင် ကန့်သတ်ထားတာပါ ။

- **Visibility Constraints –** ဒီ constraint ကတော့ link ပေါ်မှာ point တစ်ခု သတ်မှတ်ပြီး sensor(eg-camera,3d sensors) ကနေ မြင်နိုင်တဲ့ မြင်ကွင်းအတွင်းမှာပဲ ရှိနေအောင် ကန့်သတ်ထားတာပါ ။ အဓိပ္ပါယ်က sensor ကနေ link ပေါ်မှာရှိတဲ့ အဲ့ point လေးကို visible ဖြစ်နေတယ်ဆိုရင် link က သူ့ constraint ကိုလိုက်နာတယ်ပေါ့ ။

- **Joint Constraints –** ဒီ contraint ကတော့ joint limit သတ်မှတ်တာပါ ၊ joint value ကို သတ်မှတ်ထားတဲ့ value နှစ်ခုအတွင်းရှိနေအောင်ထားတာပါ ။

- **User-defined Constraints -** ဒါကတော့ user ဆီကနေ စိတ်ကြိုက် constraint သတ်မှတ်လို့ရတယ်လို့ပြောချင်တာပါ။

2.b.(iv) – Motion Plan Result

ကျနော်တို့လိုချင်တဲ့ Motion Plan Request ကို response တဲ့အနေနဲ့ move_group node ကနေ Motion Planner တွေနဲ့ ချိတ်ဆက်ပြီး အပေါ်မှာကျနော်ပြောခဲ့တဲ့ constraints တွေကိုလိုက်နာတဲ့ trajectory တစ်ခုထုတ်ပေးတာပါ ။ ပြီးတော့ အဲ့ဒီ trajectory ကို ကျနော်တို့ Manipulator ရဲ့ joint trajectory controller တွေဆီလှမ်းပို့ပြီး ကျနော်တို့ Manipulator ကို desired location(pose) ကို ရွေ့လျားစေတာပါ ။ သတိပြုရမှာက နောက်ဆုံး Result အနေနဲ့ move_group node ကနေ ထုတ်ပေးတာက Trajectory ပါ ၊ Path မဟုတ်ပါဘူး ။ အထပ်ထပ် အခါခါ သတိပေးပါတယ်နော် ။

2.b.(v) – Motion Plan Request Adapters

Motion Planning Pipeline တစ်ခု ပြီးပြည့်စုံစေဖို့ Request,Response အပြင်နောက်ထပ်လိုတာကတော့ Planning Request Adapters ပါ ။ Planning Request Adapters က Motion Plan Request တွေကို Pre-processing လုပ်ခြင်းနှင့် Motion Plan Response တွေကို Post-Processing လုပ်ပေးပါတယ် ။ Pre-Processing လုပ်တယ်ဆိုတာကို ရှင်းအောင် ဥပမာပေးရရင် မိမိတို့ Manipulator က သတ်မှတ်ထားတဲ့ Joint Limits ကို ကျော်နေတယ်ဆို သက်ဆိုင်ရာ Adapters က Joint Limit ထဲဝင်အောင် fix လုပ်ပေးမှာပါ ။Pre-Processing လုပ်တယ်ဆိုတာကတော့ ကျနော် ခုနက အပေါ်မှာသေချာလေးရှင်းပြခဲ့သလိုပဲ Motion Planner တွေကနေ Generate လုပ်လိုက်တဲ့ Path ကို Time Parameterization လုပ်ပြီး Trajectory ပြောင်းတယ်ဆိုတာကို သက်ဆိုင်ရာ Adapter က လုပ်ပေးတာပါ ။ ရှင်းပြီထင်ပါတယ် ။ အောက်က Fig (2.7) မှာ ကြည့်ပါ ။

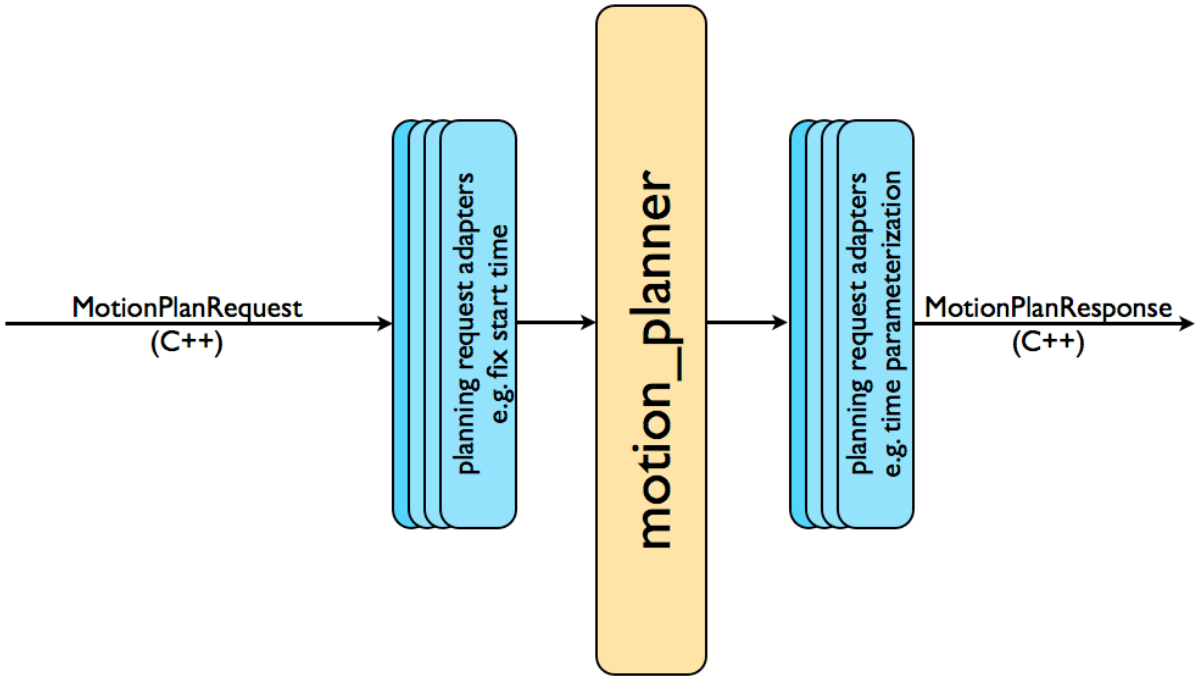


Fig – 2.7

သုံးတဲ့ Adapter ပေါ်မူတည်ပြီး Function တွေလည်းကွာသွားမှာပါ ။ Moveit မှာပါတဲ့ Default Planning Request Adapters တချို့ကို အောက်မှာပြောပြပါမယ် ။

- **FixStartStateBounds –** ဒီ Adapter က ကျနော်တို့ Manipulator ရဲ့ Initial Joint State က ကျနော်တို့ URDF ထဲမှာ ရှိတဲ့ Joint Tag ရဲ့ Sub Tag ဖြစ်တဲ့ limit tag ထဲမှာ သတ်မှတ်ခဲ့သော joint limits အတွင်း ရှိမနေရင် Initial Joint ကို Joint Limits အတွင်းဝင် အောင် Fix လုပ်ပေးပါတယ် ။ ဒါကို နားမလည်ရင် URDF ကို ပြန်လေ့လာဖို့ အကြံပြုပါတယ် ။ ဒါက အပေါ်မှာ Pre-Processing ကိုပြောတုန်းက ဥပမာ ပေးခဲ့သလိုပါပဲ ။

- **FixWorkspaceBounds –** ဘယ် Manipulator အတွက်မဆို သူအလုပ်လုပ်မယ့် Workspace ဆိုတာ သတ်မှတ်ပေးရပါတယ် ။ ဒီ Adapter က ကျနော်တို့ Manipulator တစ်ခုလုံးက Workspace အတွင်းမှာ ရှိမနေရင် သတ်မှတ်ထားတဲ့ Workspace cube

size ဖြစ်တဲ့ 10m x 10m x 10m အတွင်းရောက်အောင် Fix လုပ်ပေးပါတယ် ။ ဒါလည်း Pre-processing ပါ ။

- **FixStartStateCollision –** Manipulator ရဲ့ Joint တွေက world ထဲက object တစ်ခုခုနဲ့ collision ဖြစ်နေတာပဲဖြစ်ဖြစ် သူ့ link အချင်းချင်း self-collision ဖြစ်နေတာပဲဖြစ်ဖြစ် ဒီ Adapter က collision free ဖြစ်တဲ့ configuration space တစ်ခုကိုရှာပြီး Fix လုပ်ပေးပါတယ် ။ အဲ့လို collision ဖြစ်နေတဲ့ current configuration space ကနေ collision free ဖြစ်တဲ့ new configuration space တစ်ခုကို ပြောင်းဖို့အတွက် သူ့ Adapter မှာပါတဲ့ “jiggle_factor” parameter လေးက လုပ်ပေးပါတယ် ။ ဒါလည်း Pre-processing ပါ ။

- **FixStartStatePathConstraints –** ဒီ Adapter က ကျနော်တို့ Manipulator ရဲ့ initial Pose က သတ်မှတ်ထားတဲ့ Path Constraints အတွင်းရှိမနေခဲ့ရင် new pose တစ်ခုရှာပြီး Path Constraints အတွင်းရောက်အောင် Fix လုပ်ပေးပါတယ် ၊ ပြီးတော့ အဲ့ဒီ new pose ကို initial pose အဖြစ်သတ်မှတ်ပါတယ် ။ဒါလည်း Pre-processing ပါ

- **AddTimeParameterization -** ဒါက အပေါ်မှာ Post Processing ကိုပြောတုန်းက ဥပမာပေးခဲ့သလိုပါပဲ ။

2.c – Planning Scene

2.c.(i) – Introduction to Planning Scene

Planning Scene ဆိုတာကတော့ ကျနော်တို့ Manipulator ရှိနေမယ့် world environment ကို represent တဲ့နေရာမှာသုံးပါတယ် ၊ ပြီးတော့ သူက Manipulator ရဲ့ state ကိုလည်း store လုပ်ပါတယ် ။ ထို Planning Scene ကို move_group node ထဲမှာရှိတဲ့ Planning Scene Monitor က Handle လုပ်ပါတယ် ၊ represent လုပ်တယ်ပေါ့ဗျာ ။

2.c.(ii) – Planning Scene Monitor

Planning Scene Monitor က အောက်က သုံးခုကို listen လုပ်ပြီး Planning Scene ကို Represent လုပ်တာပါ ။

- **State Information –** robot ကနေ publish လုပ်ပေးတဲ့ joint_state topic ပါ ။
- **World Geometry Information -** User ဆီကနေ input အနေနဲ့ publish လုပ်ပေးတဲ့ planning_scene topic ပါ ။
- **Sensor Information –** အပေါ်မှာ တမင်မပြောဘဲ ချန်ခဲ့တာပါ ၊ Planning Scene Monitor ထဲမှာ World Geometry Monitor ဆိုတာလေး ပါပါတယ် ၊ အဲ့ဒီ World Geometry Monitor က sensor information တွေကို ရယူမှာပါ ၊ အောက်မှာ Topic တစ်ခုအနေနဲ့ အသေးစိတ်ဆက်ရှင်းမှာပါ ။

အောက်က Fig (2.8) ကို ကြည့်ရင် အပေါ်က သုံးချက်ကို နည်းနည်း သဘောပေါက်ပါ လိမ့်မယ် ။

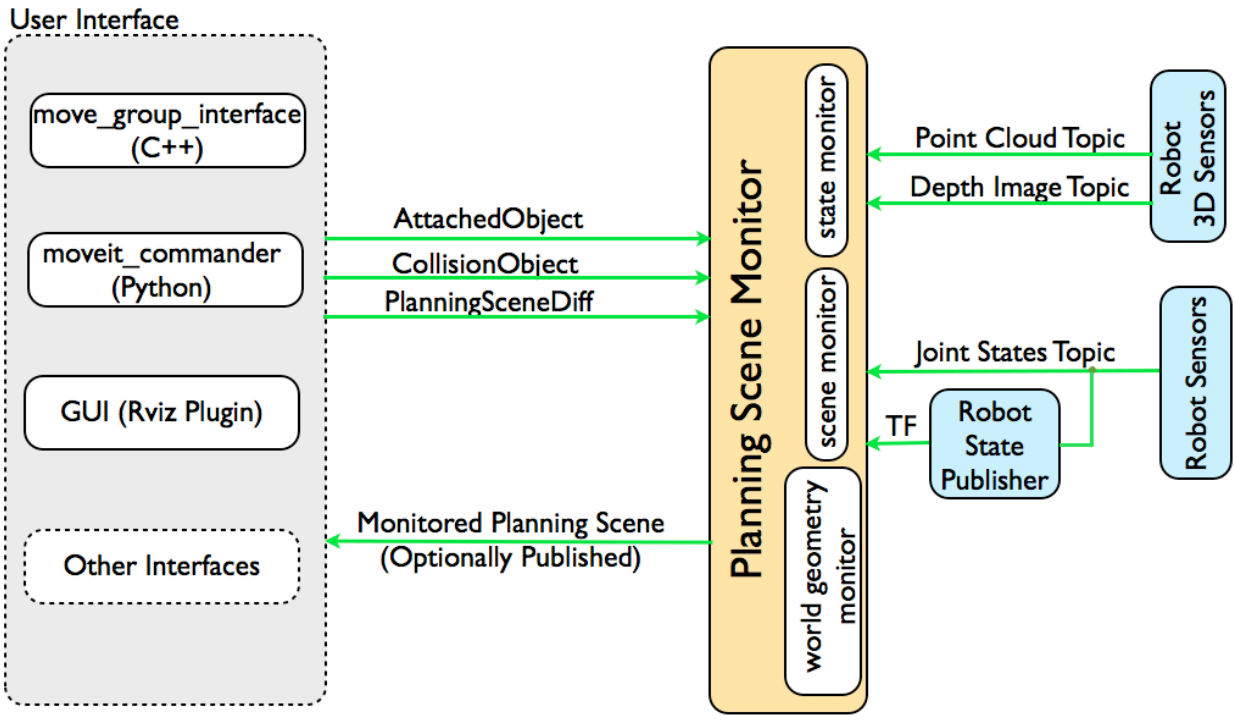


Fig – 2.8

2.c.(iii) – World Geometry Monitor

အပေါ်မှာ ပြောခဲ့တဲ့ နောက်ဆုံးနှစ်ချက်ဖြစ်တဲ့ **World Geometry Information** နဲ့ **Sensor Information** တွေကို World Geometry Monitor ကနေ ရယူအသုံးပြုပြီး World Geometry ကို build လုပ်ပေးတာပါ ။ ဘယ်လို Build လုပ်လဲဆိုတော့ World Geometry Monitor ထဲမှာ ပါတဲ့ Occupancy Map Monitor ကနေ Environment ရဲ့ 3D Representation ကို Sensor Information တွေအသုံးပြုပြီး ဖော်ပြပေးနိုင်တဲ့ Octomap ဆိုတဲ့ 3D Perception Library တစ်ခုကို အသုံးပြုပြီး Build လုပ်ပေးတာပါ ။

အဲ့ဒီ Octomap ကို MoveIt မှာ inbuilt ပါတဲ့ Plugin နှစ်ခုကနေ Generate လုပ်ပေးမှာပါ ။ အောက်က နှစ်ခုပါ ။

- **Point Cloud Occupancy Map Updater Plugin –** ဒီ Plugin က 3D Sensor ကရတဲ့ Point Cloud ဆိုတဲ့ sensor information ကို input အနေနဲ့ရယူပြီး Generate လုပ်ပေးမှာပါ ။
- **Depth Image Occupancy Map Updater Plugin –** ဒီ Plugin က 3D Sensor ကရတဲ့ Depth Image ဆိုတဲ့ sensor information ကို input အနေနဲ့ရယူပြီး Generate လုပ်ပေးမှာပါ ။

Octomap ကိုအသုံးပြုပြီး Manipulator ရှိနေမယ့် World ကို Represent လုပ်တာကို အောက်ပုံ Fig (2.9) မှာ နမူနာကြည့်နိုင်ပါတယ် ။ Octomap အကြောင်း သေချာလေ့လာချင်ရင် ဒီ Link မှာ လေ့လာလို့ရပါတယ် ။ (http://octomap.github.io/)

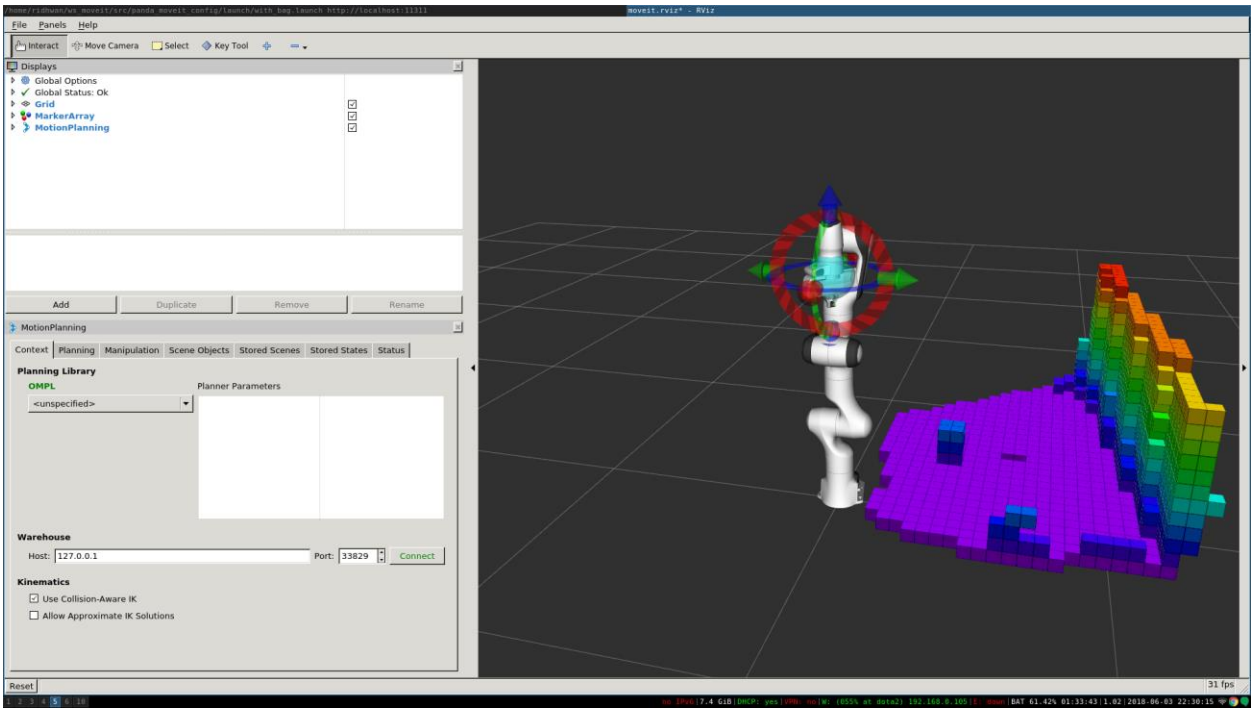


Fig – 2.9

2.d – Kinematic Plugins

MoveIt က Kinematics အတွက် Plugin တွေလည်း အဆင်သင့်ရှိပြီးသားဖြစ်သလို ကိုယ်လိုအပ်သလို Own Inverse Kinematics Plugins တွေလည်း User တွေကို ရေးနိုင်အောင် သူ့ရဲ့ infrastructure ကို လုပ်ထားပါတယ်။

Forward Kinematics နဲ့ Jacobians တွေကိုတော့ MoveIt ရဲ့ moveit core library မှာရှိတဲ့ RobotState Class ထဲမှာ အဆင်သင့် Integrate လုပ်ပြီးသားပါ ၊ ပူစရာမလိုပါဘူး ။

Inverse Kinematics (IK) အတွက် default ဖြစ်တဲ့ KDL inverse kinematics plugin ကို MoveIt Setup Assistant ကနေ automatically generate လုပ်ပေးမှာပါ ၊ အဲ့ဒီ့ KDL Default Inverse Kinematics Plugin က numerical jacobian-based solver ကိုအသုံးပြုထားတာ ဖြစ်ပါတယ် ။numerical solver

ဖြစ်လို့ IK ကို calculate လုပ်ရာမှာ အချိန်ပိုကြာနိုင်ပါတယ် ၊ အဲ့ဒါကြောင့် Analytical method ကိုသုံးတဲ့ IKFast Solver ဆိုတဲ့ Package ကိုလည်းအသုံးပြုလို့ရပါတယ် ။ ဒါပေမယ့် သတိပြုရမှာက IKFast က DOF 6 ခုထက်နည်းတဲ့ Manipulator ကိုသုံးမှ ပိုပြီး အချိန်တိုတိုအတွင်း perform လုပ်နိုင်မှာပါ ။

ဒီ ခေါင်းစဉ်မှာသုံးသွားတဲ့ Forward Kinematics နဲ့ Inverse Kinematics တွေကို မသိသေးရင် သူတို့အကြောင်းကို အရင်လေ့လာပါလို့ တိုက်တွန်းချင်ပါတယ် ၊ MoveIt ကို လေ့လာနေမှတော့ အဲ့ဒါတွေကို သိပြီးသားလို့ ယူဆပါတယ် ။

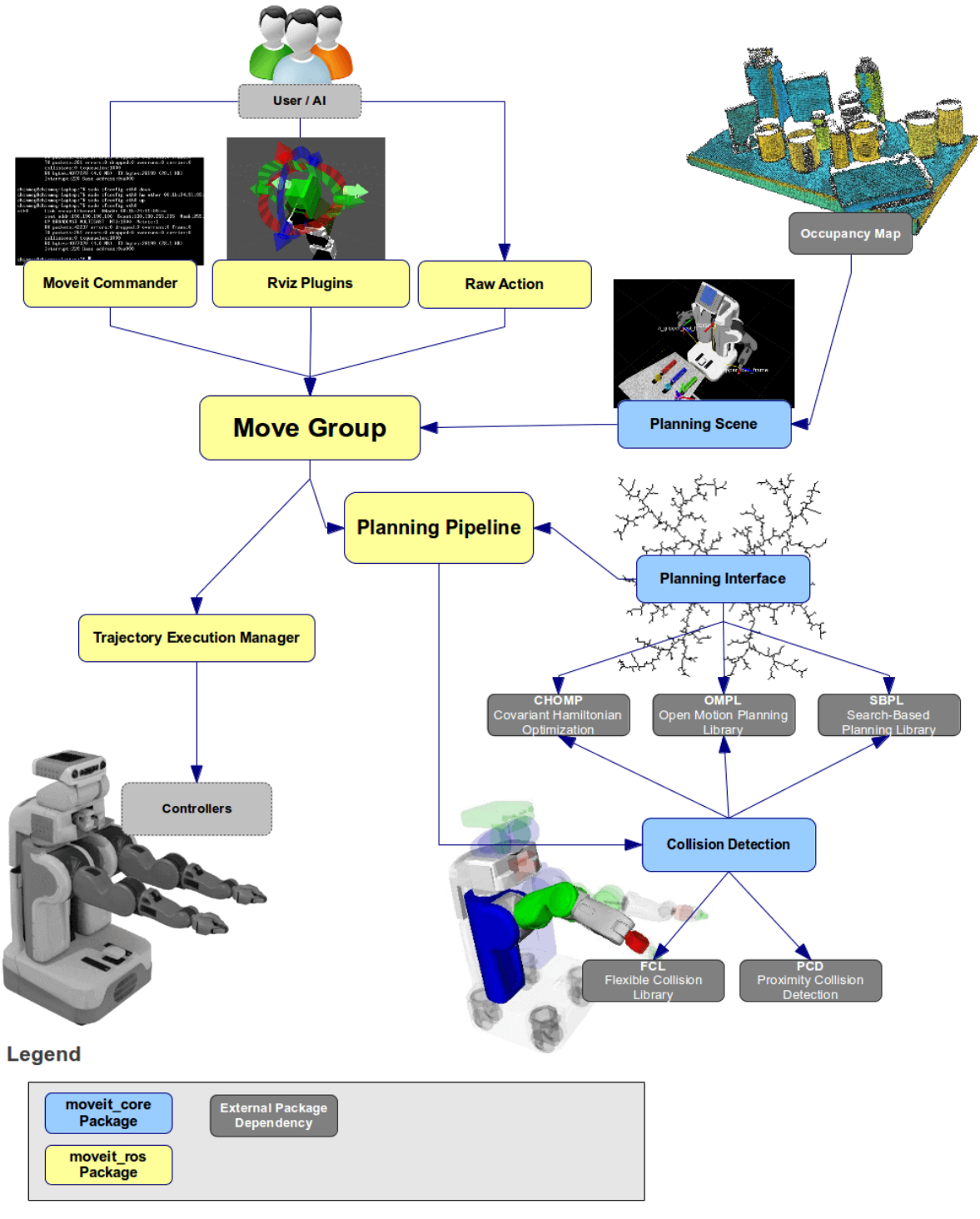
2.e – Collision Checking

Collision Checking ကတော့ အပေါ်မှာလည်း ခဏခဏပါပြီးသားဖြစ်လို့ ရင်းနှီးပြီးသား ဖြစ်မှာပါ ။ Collision Checking ကို Planning Scene ထဲမှာ "CollisionWorld" ဆိုတဲ့ object တစ်ခုကို အသုံးပြုပြီး Configure လုပ်တာပါ ။ MoveIt မှာ အောက်ပါ collision object သုံးခုကို collision checking လုပ်လို့ရပါတယ် ။

- **Meshes**
- **Primitive Shapes (Eg -** boxes, cylinders, cones, spheres and planes)
- **Octomap Object**

Collision Checking လုပ်ခြင်းသည် computationally အရ expensive task တစ်ခုဖြစ်ပါတယ် ။ ထို့ကြောင့် အဲ့ဒီ computation ကို လျော့ချဖို့အတွက် **ACM**(Allowed Collision Matrix) ဆိုတဲ့ matrix တစ်ခုကို MoveIt က provide လုပ်ပေးပြီးသားပါ **။ACM** ဆိုတာ ကျနော်တို့ collision checking အတွက် binary value နဲ့ယူပြီး တွက်ချက်ပေးတာပါ ၊ ဥပမာပေးရမယ်ဆို ကျနော်တို့ Manipulator နဲ့ World ကြား collision checking လုပ်ကြမယ်ဆိုပါစို့ ၊ အကယ်၍ ကျနော်တို့ Manipulator နဲ့ World ကြားတန်ဖိုးကို **ACM** ထဲမှာ binary value 1 လို့သတ်မှတ်ခဲ့မယ်ဆို အဲ့ဒီ Manipulator နဲ့ World ကြား collision checking လုပ်ဖို့မလိုဘူးဆိုပီး သတ်မှတ်လို့ရပါတယ် ၊ အဲ့ဒီလို မလိုအပ်တဲ့ bodies တွေကြား collision checking မလုပ်ခြင်းဖြင့် computation အရ အများကြီးသက်သာသွားပါတယ် ။ ဟိုးအပေါ်မှာ လည်း တစ်ခေါက်ပြောခဲ့ပီးပီ ဖြစ်ပါတယ် ၊ SRDF ဆိုတဲ့ ခေါင်းစဉ်အောက်မှာပါ ။

ကျနော်တို့ Fig (2.1) မှာ အမြည်းကျွေးတယ်လို့ ပြောခဲ့ပါတယ် ၊ ခုက Moveit Concept တစ်ခုလုံးကို သေချာလေး အသေးစိတ်ရှင်းပြီးပြီဖြစ်လို့ နားလည်သွားရင် အောက်က ပုံက သင့်ကို ပြီးပြည့်စုံစေမှာပါ ၊ Fig (2.1) က ပုံပါပဲ ။



REFERENCE

1. MOVEIT-ROS OFFICIAL DOCUMENTATION
2. Mastering ROS For Robotics Programming PDF

Credit to MOVEIT-ROS OFFICIAL DOCUMENTATON FOR IMAGES USED IN THIS PDF

----------END----------