

机器人动作重定向

技术文档

项目名称： 人形机器人动作重定向系统
作者： 沈玉祥
日期： 2026 年 4 月 6 日
版本： v1.0

目录

1	程序文件说明	3
1.1	概述	3
1.2	核心文件	3
1.3	服务器端文件	4
1.4	其他说明	7
2	重定向方法与原理	8
2.1	常见的三种重定向方法	8
2.2	整体流程	8
2.3	坐标系处理	9
2.3.1	SMPL 坐标系与机器人坐标系的对齐	9
2.3.2	朝向对齐预处理	10
2.4	骨骼重定向	10
2.4.1	动态 Anchor 设计	10
2.4.2	链式骨骼长度替换	11
2.5	差分 IK 求解	12
2.5.1	PINK 框架介绍	12
2.5.2	任务设计与权重	12
2.5.3	躯干朝向约束	13
2.5.4	头部朝向处理	13
2.6	数据处理与平滑	13
2.6.1	插值扩帧	13
2.6.2	加权滑动滤波	13
2.6.3	关节限位约束	14
2.6.4	脚踝角度计算	14
2.7	其他	14
2.7.1	遥操作与实时重定向的区别	14
2.7.2	局部四肢避障方法	15
3	常见踩坑	16
3.1	骨骼重定向相关	16
3.2	IK 求解相关	16
4	后续规划	18
4.1	目标	18
4.2	计划路线	18
4.3	参考方法	18

A 关键参数说明

19

1 程序文件说明

1.1 概述

主要包括：人体动作到机器人动作的重定向方法、动作之间的平滑插值、视频提取到机器人动作的方法等内容。

1.2 核心文件

ik_redirection_npy.py

功能：将人体动作信息转化为机器人各关节角度 (rad) 信息。

主要类/函数：

- H1Config —— 机器人配置信息
- H1PinkSolver —— 核心求解器
- process_motion() —— 主处理函数
- apply_bone_retargeting_single() —— 单帧骨骼重定向
- load_data_all_npy() —— 输入格式转换： $[N, 22, 3, T] \rightarrow$ 第 i 个 $[T, 22, 3]$

输入： $[N, 22, 3]$ 的 numpy 数组， N 为帧数，22 为 SMPL 主要关节数，3 为 XYZ 坐标

输出：字典，主要为 dof（关节角度序列，弧度，shape 为 $[N, 28]$ ）

运行效率：0.08s/196 帧 $\approx$ 0.4ms/帧（单线程，CPU）

easy_MotionInterpolator.py

功能：实现机器人动作（关节角度 dof）的插值、帧率重定向及平滑处理；可处理过渡的自碰撞问题。

主要配置项：

- MotionInterpolator —— 核心插值类
- interpolate() —— 对 dof 序列进行三次样条插值
- smooth() —— 加权滑动滤波消除插值引入的抖动
- process() —— 依次执行插值与平滑，返回处理后的 dof 序列

输入：低帧率的关节角度序列，shape 为 $[N, 28]$

输出： 高帧率平滑后的关节角度序列，shape 为 $[N', 28]$

运行效率： 单次拼接（含碰撞检测） $\approx 0.8\text{ms}$ （单线程，CPU）

 vis.py

功能： 逐帧播放并可视化。

主要配置项：

- load_motion_data() —— 加载动作数据
- build_obstacle_geoms() —— 根据障碍物数据构建红色线框盒几何体
- main() —— 逐帧绘制机器人关节角度与根节点状态

输入： 包含 dof 序列、根节点位姿、可选的 h1_joint_pos 和 smpl_joints_target

输出： Isaac Gym 实时可视化窗口

依赖： Isaac Gym、PyTorch（支持 GPU 加速渲染）

1.3 服务器端文件

位置： 服务器下的 GVHMR 目录下；conda activate gvhmr

 GVHMR/run.py

功能： 把人体动作视频 (mp4) 转换成机器人各关节电机角度 (rad)。

主要配置项：

- GVHMRSystem —— 核心系统类
- __init__() —— 冷启动初始化，加载 GVHMR 主模型、Tracker、VitPose、特征提取器、SMPL 模型及 H1PinkSolver
- process_video() —— 热运行入口，处理单个视频，依次执行预处理、GVHMR 推理、H1 IK 优化，返回关节角度结果
- _run_preprocess_efficient() —— SLAM 后台线程并行，Tracker、VitPose、特征提取串行 GPU 推理
- _run_h1_optimization() —— 从 GVHMR 输出的 SMPL 参数提取关节坐标，调用 H1PinkSolver 完成重定向

输入： mp4 视频文件

输出： 包含 dof 关节角度序列 (rad, shape 为 $[N, 28]$) 的字典

使用方法：

```

1 conda activate gvhr
2 cd GVHMR/
3 python run.py --input (视频路径)
4 # 结果存储到 outputs 目录的 result.pkl 文件

```

运行效率：4.5s/8s (4090GPU)

 GVHMR/new-server.py

功能：支持大模型工具调用的服务端。

主要配置项：

- `_run_gvhr()` ——核心推理函数
- `_frames_to_tmp_video()` ——将帧列表写入临时视频文件
- `_process_frames_async()` ——异步推理包装

主要接口：

- GET /tools ——查询当前可用工具，大模型调用前先请求：

```
1 curl http://localhost:8003/tools
```

- POST /tool/call ——大模型调用工具的入口，传工具名和参数：

```

1 curl -X POST http://localhost:8003/tool/call \
2     -d '{"name": "process_video_to_robot_motion",
3         "parameters": {"video_path": "/data/demo.mp4"}}'

```

- POST /generate ——直接输入视频文件路径处理：

```

1 curl -X POST http://localhost:8003/generate \
2     -d '{"video_path": "/data/demo.mp4"}'

```

- WS /ws/stream ——接入实时相机推流，配合 `camera_client.py` 使用：

```
1 python camera_client.py --camera 0
```

输入：视频文件绝对路径，或通过 WebSocket 推送的 JPEG 编码帧字节流

输出：字典，主要为 `dof` (关节角度序列, 弧度, shape 为 $[N, 28]$)、`fps`、`duration_sec`

 GVHMR/camera_client.py

功能：获取本地摄像头，将视频帧实时推送到 new-server.py 进行推理，并接收返回的关节角度。

主要函数：

- run_camera_stream() ——打开摄像头、推帧、接收结果
- _handle_result() ——处理每次推理返回的结果

使用方法：

```
1 # 默认摄像头，默认参数
2 python camera_client.py
3
4 # 指定摄像头编号
5 python camera_client.py --camera 0 --fps 30 --chunk_sec 4
6
7 # 服务端不在同一台机器时指定地址
8 python camera_client.py --server 192.168.1.100:8003
9
10 # 固定机位相机
11 python camera_client.py --static_cam
```

输入：本地摄像头实时画面（通过 OpenCV 读取）

输出：每隔 chunk_sec 秒打印一次推理结果，格式如下：

```
1 [推理结果] chunk_id=0 | 帧数=120 | 时长=4.0s | fps=90 | DOF shape
   =[360, 28]
```

备注：_handle_result() 函数内的注释处（第 117 行）可对接机器人控制器，将 DOF 数据发给 ROS 节点。

 GVHMR/test_stream.py

功能：用本地 mp4 视频文件模拟摄像头推流，测试 new-server.py 的视频流接口是否正常工作。

主要函数：

- test_stream() ——读取视频文件逐帧推送到服务端，并接收推理结果
- _print_result() ——打印每个 chunk 的推理结果

使用方法：

```
1 # 基本用法（必须指定视频文件）
2 python test_stream.py --video /data/demo.mp4
3
4 # 指定参数
5 python test_stream.py --video /data/demo.mp4 --fps 30 --chunk_sec 4
6
7 # 服务端不在同一台机器时指定地址
8 python test_stream.py --video /data/demo.mp4 --server
   192.168.1.100:8003
```

输入：本地视频文件路径

输出：每积累 `chunk_sec` 秒的帧触发一次推理，打印结果：

```
1 [chunk 0] dof_shape=[360, 28], fps=90, duration=4.0s
2 [done] 共处理 2 个 chunk
```

1.4 其他说明

- **根节点固定：**本模块的目标是从人体动作生成机器人电机角度序列，机器人 waist 点在世界坐标系中保持固定，相当于将机器人腰部悬挂于空间中。根节点的全局位移不参与重定向计算。
- **输出崩溃排查：**若输出的电机角度出现突变或崩溃，大概率是输入动作幅度过大，超出机器人关节限位范围，导致 IK 无法在单帧内追上目标。解决方法是适当降低大模型生成的动作幅度，或延长动作时长使运动更平缓。
- **体型通用性：**重定向模块对输入人体的臂长、身高、体型无要求，骨骼重定向阶段会自动将 SMPL 骨骼方向映射到机器人实际骨骼长度，不受输入人体比例影响。

2 重定向方法与原理

2.1 常见的三种重定向方法

常见的机器人动作重定向方法主要分为三类：梯度下降方法、神经网络模型 +IK 微调、纯 IK 逆解方法三种。

表 1: 三种重定向方法对比

方法	优点	缺点	适用场合
梯度下降	动作质量高；批量处理性价比好	实时性差，处理数秒视频需 2s 以上；需预先标定缩放比例	离线批量处理
神经网络 + IK 微调	推理速度快；兼顾数据驱动与几何约束	依赖大量高质量标注数据，准备成本高；对体型变化鲁棒性弱	数据充足的生产环境
纯 IK 逆解 (本文)	无需训练数据；实时求解；体型适应性好	需合理设计任务约束与权重	实时重定向

2.2 整体流程

本系统的重定向流程如下：



2.3 坐标系处理

2.3.1 SMPL 坐标系与机器人坐标系的对齐

SMPL 以骨盆 (pelvis) 为根节点, 机器人以 waist 为躯干中心, 两者物理意义最接近, 因此将 pelvis 对齐到机器人 waist 位置。具体分两步: 以 pelvis 为原点中心化后乘以缩放系数, 再平移至机器人 waist 高度:

```

1 # 中心化 + 缩放
2 smpl_joints = (smpl_raw_seq - smpl_raw_seq[:, 0:1, :]) * self.SMPL_SCALE
3 # 平移至机器人waist高度
4 waist_pos = self.configuration.get_transform_frame_to_world("waist").
    translation
5 smpl_joints += waist_pos.reshape(1, 1, 3)

```

2.3.2 朝向对齐预处理

大模型输出的动作可能包含根节点移动（如走圈），直接重定向会导致机器人跟随转向。因此对每帧计算髋部左右向量的水平偏转角，绕 Z 轴旋转补偿，使全程朝向与第 0 帧保持一致：

```

1 # 计算每帧髋部向量的yaw角
2 hip_vecs = smpl_joints[:, idx_r] - smpl_joints[:, idx_l]
3 hip_vecs[:, 2] = 0 # 投影到水平面
4 yaw_ref = np.arctan2(hip_vecs[0, 1], hip_vecs[0, 0])
5 yaw_all = np.arctan2(hip_vecs[:, 1], hip_vecs[:, 0])
6
7 # 绕Z轴旋转补偿，固定朝向
8 rot_mats = sRot.from_euler('z', yaw_ref - yaw_all).as_matrix()
9 centered = smpl_joints - smpl_joints[:, 0:1, :]
10 smpl_joints = np.einsum('nij,nkj->nki', rot_mats, centered) + smpl_joints
   [:, 0:1, :]

```

2.4 骨骼重定向

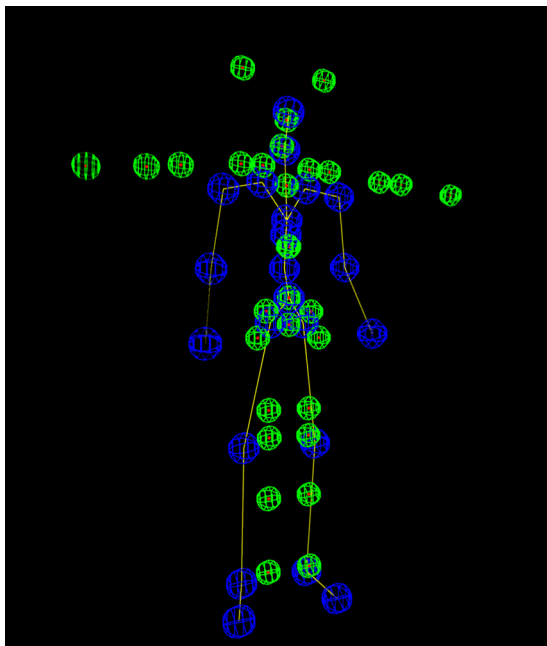


图 1: 重定向前

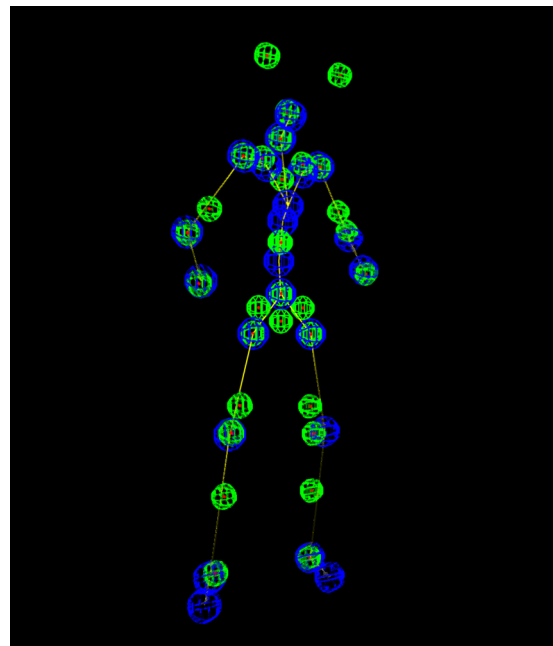


图 2: 重定向后

图 3: 重定向前后对比：绿色为 H1 机器人关节点，蓝色为 SMPL 目标关节点

2.4.1 动态 Anchor 设计

SMPL 人体与机器人结构不同，肩部和髋部的关节位置无法直接对应。若直接以 SMPL 的肩膀点作为 anchor，会有两个问题：一是两者骨骼比例不同导致位置偏差；二

是人体上肢挥动时 SMPL 肩膀点存在上下位移，而机器人肩部三个电机的结构无法复现这种位移。

因此每帧从机器人当前正运动学结果中读取肩部、髋部、颈部的真实世界坐标作为 anchor，再以 SMPL 骨骼的方向信息驱动末端位置：

```
1 # 每帧从机器人正运动学读取 anchor 点
2 anchors = {
3     'l_shoulder': self.configuration.get_transform_frame_to_world(
4         "left_upper_arm").translation.copy(),
5     'r_shoulder': self.configuration.get_transform_frame_to_world(
6         "right_upper_arm").translation.copy(),
7     'l_thigh':    self.configuration.get_transform_frame_to_world(
8         "left_thigh").translation.copy(),
9     'r_thigh':    self.configuration.get_transform_frame_to_world(
10        "right_thigh").translation.copy(),
11     'neck':       self.configuration.get_transform_frame_to_world(
12         "neck_linkage").translation.copy(),
13 }
```

同时，利用 SMPL 的 spine1、left_collar、right_collar 三点构造正交基，得到躯干朝向，用于约束机器人胸部旋转，间接驱动上肢姿态跟随躯干运动：

```
1 # 三点构造胸部正交基
2 up      = _normalize((p_l_collar + p_r_collar) / 2.0 - p_spine3)
3 right    = _normalize(p_l_collar - p_r_collar)
4 forward  = _normalize(np.cross(right, up))
5 right    = _normalize(np.cross(up, forward))
6 chest_rot = np.column_stack([forward, right, up])
```

2.4.2 链式骨骼长度替换

SMPL 与机器人的骨骼长度不同，直接使用 SMPL 关节坐标会导致末端位置偏差。解决方案是保留 SMPL 各骨骼段的方向向量，将长度替换为机器人实际物理尺寸，从 anchor 点出发链式计算各关节位置，使末端（手、脚）能够正确对齐：

```
1 # 机器人实际骨骼长度（米）
2 ROBOT_DIMS = {
3     'thigh':      0.0832,    # 髋 -> 膝
4     'calf':       0.1105,    # 膝 -> 踝
5     'upper_arm':  0.07579,   # 肩 -> 肘
6     'forearm':    0.04739,   # 肘 -> 腕
7     'neck2head':  0.022487329, # 颈部 -> 头
8 }
9
10 # 以 anchor 为父节点，沿 SMPL 方向链式扩展（以左腿为例）
11 nj[:, b['left_hip']] = anchors['l_thigh']
12 nj[:, b['left_knee']] = anchors['l_thigh'] \
```

```

13     + _normalize(j[:, b['left_knee']] - j[:, b['left_hip']]) * d['thigh']
14 nj[:, b['left_ankle']] = nj[:, b['left_knee']] \
15     + _normalize(j[:, b['left_ankle']] - j[:, b['left_knee']]) * d['calf']

```

左右臂同理，从肩部 anchor 出发，依次计算肘、腕位置。经过链式替换后，末端关节坐标既保留了 SMPL 动作的姿态方向，又符合机器人的实际骨骼比例。

2.5 差分 IK 求解

2.5.1 PINK 框架介绍

PINK (Python Inverse kiNematics) 是基于 Pinocchio 的差分 IK 框架。Pinocchio 负责机器人运动学/动力学计算，PINK 在其基础上将 IK 问题构造为二次规划 (QP) 问题，通过定义多个带权重的**任务** (Task) 和**约束** (Limit) 来描述求解目标。

每个任务对应一个末端执行器的位置或旋转目标，权重决定各任务的优先级。求解器在满足关节限位约束的前提下，每帧求解一个关节速度增量，再积分更新机器人配置，从而逐帧追踪目标轨迹。本系统使用 quadprog 作为底层 QP 求解器，阻尼系数设为 10^{-3} 防止奇异。

2.5.2 任务设计与权重

系统共定义以下 IK 任务，各任务的位置权重与旋转权重如表所示：

表 2: IK 任务权重配置

机器人 Link	对应 SMPL 关节	位置权重	旋转权重
chest	—	0.0	50.0
left_hand / right_hand	left_wrist / right_wrist	10.0	0.0
left_foot / right_foot	left_ankle / right_ankle	10.0	1.0
left_force_arm / right_force_arm	left_elbow / right_elbow	5.0	0.0
left_calf / right_calf	left_knee / right_knee	5.0	0.0
head	head	1.0	5.0

chest 任务不追位置，只追旋转，用于约束躯干朝向防止后仰；手和脚作为末端执行器位置权重最高；肘和膝作为中间关节权重较低，主要用于引导肢体方向；头部以旋转约束为主。此外还加入了一个低权重 (0.1) 的 PostureTask 作为软约束，防止未被追踪的关节乱飘：

```

1 self.posture_task = pink.tasks.PostureTask(self.model)
2 self.posture_task.set_target(self.q_ref)
3 self.posture_task.cost = 0.1

```

2.5.3 躯干朝向约束

IK 求解时若不对躯干朝向加以约束，机器人上半身容易出现后仰或侧倾。为此单独设置一个 chest 的 FrameTask，只追旋转不追位置（位置权重为 0，旋转权重为 50），每帧从 SMPL 的 spine1、left_collar、right_collar 三点构造正交基作为目标旋转矩阵（见 2.4.1 节），强制机器人胸部朝向与 SMPL 躯干保持一致。

由于位置不强制追踪，chest 的世界坐标始终取机器人当前真实位置，避免与其他任务产生冲突：

```
1 chest_pos = self.configuration.get_transform_frame_to_world("chest").
    translation.copy()
2 self.chest_task.set_target(pin.SE3(chest_rot, chest_pos))
```

2.5.4 头部朝向处理

头部朝向通过两种方式处理。若上游提供了 head_rot_mats（局部旋转矩阵），则将其与当前帧的躯干旋转矩阵相乘得到头部世界朝向；若未提供则只追位置，旋转使用单位矩阵：

```
1 if smpl_name == "head" and head_rot_mats is not None:
2     # 世界朝向 = 躯干朝向 x 头部局部旋转
3     task.set_target(pin.SE3(chest_rot @ head_rot_mats[i], limited_targets[
        smpl_name]))
4 else:
5     task.set_target(pin.SE3(np.eye(3), limited_targets[smpl_name]))
```

由于 head_rot_mats 是相对于父关节的局部旋转，对于包含大幅根节点移动的动作（如走圈），预处理阶段会自动检测并清除旋转向量的 Yaw 分量，避免头部跟随身体转向产生突变。

2.6 数据处理与平滑

2.6.1 插值扩帧

IK 求解在 20fps 上运行，完成后对关节角度序列做三次样条插值升采样至 90fps。在关节空间插值比在笛卡尔空间插值更物理合理，且计算开销小：

```
1 interp = interp1d(t_orig, sim_dof, axis=0, kind='cubic')
2 sim_dof = interp(t_new)
```

2.6.2 加权滑动滤波

对 IK 输出的关节角度序列做线性加权滑动均值滤波（窗口 7 帧），越近的帧权重越高，有效抑制 IK 求解引入的高频抖动。该滤波方法在机器人动作重定向领域被广泛采用：

```

1 w = np.arange(1, window_size + 1, dtype=float)
2 w /= w.sum() # 线性权重归一化, 越新的帧权重越大

```

2.6.3 关节限位约束

在 IK 求解时加入 ConfigurationLimit 约束, 将 28 个关节的角度限制在物理允许范围内, 防止求解结果超出机器人实际关节行程:

```

1 # 将限位写入Pinocchio模型
2 self.model.lowerPositionLimit[q_idx] = H1Config.JOINT_LIMITS[idx, 0]
3 self.model.upperPositionLimit[q_idx] = H1Config.JOINT_LIMITS[idx, 1]
4 self.config_limit = ConfigurationLimit(self.model)
5
6 # IK求解时传入约束
7 velocity = pink.solve_ik(
8     self.configuration, self.tasks, dt,
9     solver="quadprog", damping=1e-3,
10    limits=[self.config_limit],
11 )

```

2.6.4 脚踝角度计算

机器人没有脚掌关节, 脚踝 pitch 角度无法通过 IK 直接求解。因此从 SMPL 几何信息中提取每帧小腿方向向量与脚掌方向向量的夹角, 以第 0 帧 (站立姿态) 为基准计算相对变化量, 直接覆写到脚踝 pitch 电机:

```

1 # 计算小腿与脚掌方向向量的夹角
2 shin = _normalize(ankle - knee)
3 toes = _normalize(foot - ankle)
4 angle = np.arccos(np.clip((shin * toes).sum(axis=-1), -1.0, 1.0))
5
6 # 以第0帧为中性角基准, 限制在关节限位内
7 pitch = angle - angle[0]
8 pitch = np.clip(pitch, lo, hi)

```

2.7 其他

2.7.1 遥操作与实时重定向的区别

目前业内已有端到端的实时遥操作方案 (如 GR-1、AnyTeleop 等), 直接从摄像头到机器人控制信号, 延迟极低。本系统与其定位不同, 对比如下:

本系统优先接入大模型生成的 npy 动作数据, 以通用 IK 重定向为核心, 机器人结构发生变化时只需更新 URDF 和骨骼长度参数即可适配, 无需重新训练任何模型, 维护成本低。

表 3: 实时遥操作方案与本系统对比

	实时遥操作 (如 GMR)	本系统 (GVHMR + IK)
输入来源	实时摄像头	视频文件或大模型 npy 输出
实时性	极低延迟, 在线处理	离线为主, 单帧 $\approx 0.4\text{ms}$
精度	人体捕捉精度较低, 但足够还原动作	GVHMR 精度较高
机器人泛化性	换机器人需重新训练模型	只需修改 URDF 和骨骼参数, 无需重训
适用场景	实时控制、遥操作	动作生成、离线批量处理
主要优势	延迟低、部署简单	通用性强, 兼容多种输入源

2.7.2 局部四肢避障方法

在动作模仿过程中, 若摄像头检测到障碍物与机器人四肢存在碰撞风险, 需要对目标关节点位进行实时修正。

基本思路: 不修改 IK 框架, 而是在骨骼重定向阶段对 SMPL 关节点位进行几何修正, 将避障后的新点位作为 IK 追踪目标传入。

几何避障原理: 以手臂为例, 肩膀、肘部、手腕三点构成一个两边长度固定 (大臂长、小臂长) 的运动链。当大臂或小臂检测到与障碍物碰撞时, 在保持骨骼长度不变的约束下, 迭代计算绕障后的新关节位置, 使整个手臂绕过障碍物的同时保持人体物理结构不变形。腿部同理, 由髋、膝、踝三点构成固定长度运动链做同样处理。

局限性: 该方法基于纯几何计算, 时效性好, 适用于简单的局部避障场景。对于需要腰部扭转、重心转移等全身协调配合的复杂避障情况, 单纯修正四肢点位无法保证整体姿态合理性, 需要引入更完整的全身运动规划方案。

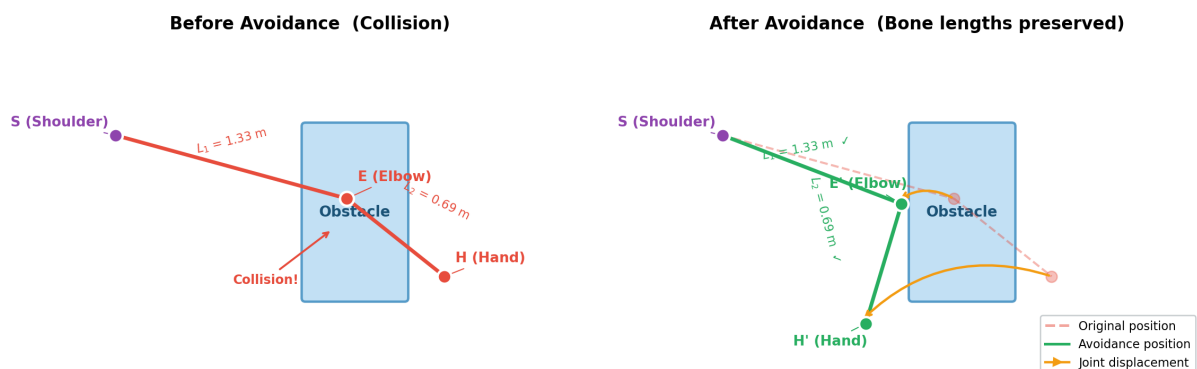


图 4: 局部四肢避障几何方法示意图

3 常见踩坑

Q: quat 旋转的作用是什么？

大模型输出的 SMPL 关节坐标以 Y 轴朝上为标准坐标系，而本系统机器人重定向以 Z 轴朝上为基准。若不做坐标系变换，输入的 SMPL 人体在机器人坐标系下呈躺倒状态，无法正常重定向。

通过四元数 $q = [0.5, 0.5, 0.5, 0.5]$ 对应的旋转矩阵，将 Y 轴朝上变换为 Z 轴朝上：

```
1 rot = sRot.from_quat([0.5, 0.5, 0.5, 0.5]).as_matrix()  
2 smpl_raw_seq = target_motion @ rot.T # [T, J, 3]
```

对于 GVHMR 视频提取的 SMPL 数据，输出已是 Z 轴朝上，无需做 Y 轴到 Z 轴的变换，只需绕 Z 轴旋转 -90° 调整朝向，对应 quat 参数为 $[0, 0, -0.707, 0.707]$ (参考服务器端 GVHMR 目录下的逆解文件)。

3.1 骨骼重定向相关

Q: 若不将躯干与两臂任务解耦会有什么后果？

若直接以 SMPL 肩膀点作为两臂 IK 追踪的 anchor，由于机器人与 SMPL 骨骼比例不同，肩膀点无法完全对齐，导致以肩膀为起点的手肘、手腕等末端追踪点整体偏移，IK 始终存在残差。

更严重的问题是，PINK 本质上求解的是 QP 问题，当多个任务存在冲突时，求解器会在各任务间做权重妥协。若躯干与手臂任务耦合，某一侧手臂恰好能追上目标而另一侧追不上时，QP 求解器为了整体最优会将误差集中到某个关节，导致该关节的轴向电机出现突变甚至疯转。

通过动态 Anchor 设计，每帧以机器人自身正运动学的肩膀位置为起点，将躯干运动与手臂末端追踪解耦，从根本上避免了上述问题。

3.2 IK 求解相关

Q: 为什么不先插值到 90fps 再进行 IK 求解？

IK 求解的耗时与帧数成正比，若先插值到 90fps 再求解，计算量是当前方案的 4.5 倍。而先在 20fps 上完成 IK 求解，再对关节角度序列做三次样条插值升采样至 90fps，精度损失极小，因为相邻帧间的关节角度变化平滑，插值能够很好地还原中间帧。

因此选择先 IK 后插值的方案，在几乎不影响动作质量的前提下将 IK 阶段耗时降低至原来的 $\frac{1}{4.5}$ ，是性价比最高的做法。

Q: 为什么不在 IK 求解过程中加入速度控制？

主要有三个原因：

第一，职责解耦。重定向模块只负责将人体动作映射为机器人关节角度，速度控制属于运动控制层的职责，两者混合会导致模块边界模糊，出现问题时难以定位。

第二，仿真与现实存在差距。PINK 求解的是仿真环境下的机器人状态，加入速度控制后轨迹会产生延迟跟踪效果，而这个延迟在真实机器人上的表现与仿真并不一致，反而引入额外误差。

第三，需求已满足。本系统的目标是离线生成某段动作对应的机器人关节角度序列，并不需要在线实时跟踪，因此静态的角度序列输出已经完全满足需求，无需引入速度控制增加复杂度。

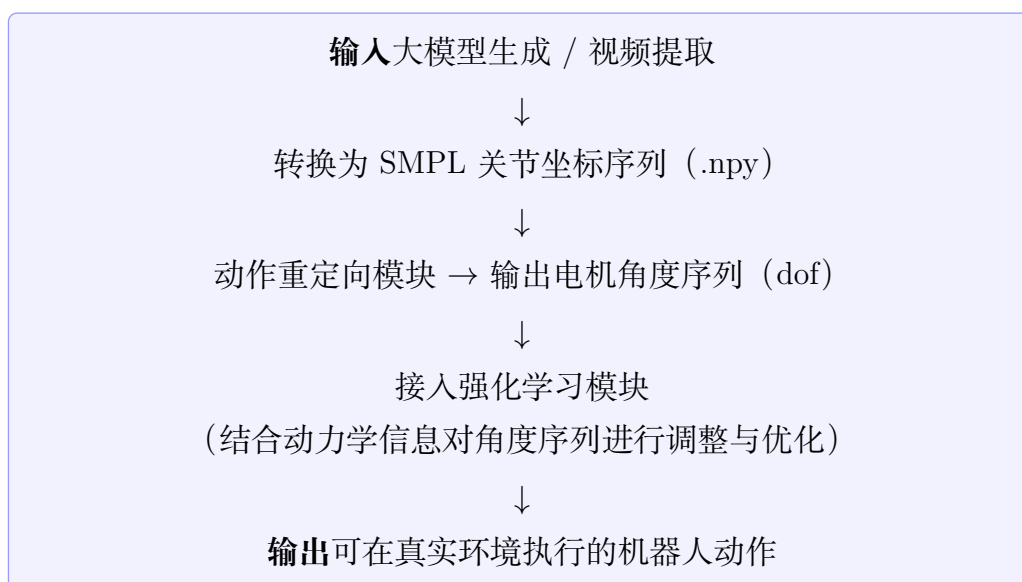
4 后续规划

4.1 目标

- **训练机器人行走：**利用重定向模块生成的高质量行走动作序列作为参考轨迹，通过强化学习训练机器人在仿真环境中实现稳定行走。
- **去掉固定根节点限制：**当前重定向模块输出的电机角度序列以固定 waist 为前提，无法直接在真实环境中执行带位移的动作。目标是结合强化学习，训练机器人在真实动力学约束下完成动作模仿，使其能够在现实环境中真正执行出对应动作。

4.2 计划路线

当前调研的技术路线如下：



4.3 参考方法

- **humanoid-gym：**基于 PPO 的 H1 行走训练，用于验证仿真环境与机器人配置。
- **ExBody2：**基于 AMP 框架的动作模仿，以重定向模块输出的 dof 序列作为参考轨迹，对重定向误差容忍度较高。
- **OmniH2O：**全身精确动作模仿备选方案，关节角度直接跟踪，还原度更高但对重定向精度要求更高。

A 关键参数说明

表 4: 重定向关键参数

参数名	值	说明
scale	0.2466	SMPL 到机器人的缩放比例
aim_fps	90	目标帧率
substeps	2	IK 每帧子步数
max_step	0.01	目标点每步最大移动距离 (m)
damping	1e-3	IK 阻尼系数