دانشگاه صنعتی شریف

دانشکده‌ی مهندسی کامپیوتر

پایان‌نامه‌ی کارشناسی

گرایش سخت‌افزار

ایجاد قابلیت راه رفتن در روبات دوپا با استفاده از روش یادگیری فعال (ALM)

پوژن ضیائی (دانشجوی لیسانس)

استاد راهنما:

دکتر سعید باقری شورکی (استادیار)

تابستان ۱۳۸۴

چکیده

روش یادگیری فعال، روش جدیدی برای مدلسازی و کنترل سیستمهای چند ورودی- یک خروجی است. اساس این روش بر تشخیص رفتارهای کلی یک سیستم به جای ذخیره کردن اطلاعات جزئی استوار است. هدف از این پروژه تولید خودکارِ کنترل‌گر روباتِ دوپا با استفاده از روش یادگیری فعال می‌باشد. این کنترل‌گر برای مدل کردن رفتار یک روبات دوپا که سیستم بسیار پیچیده و چند ورودی-چند خروجی است طراحی می‌شود به گونه‌ای که کلیه ابزارهای کنترلی با استفاده از داده های اولیه موجود به صورت خودکار تولید شوند.

کلمات کلیدی: یادگیری فعال، روبات دوپا، کنترل، مدل‌سازی

فصل اول: کلیات پروژه

فهرست

مقدمه

سالهاست که انسان سودای تولید موجودی با قابلیتهای انسانی را در سر می پروراند. آرزویی که به مرور زمان و با پیشرفت علم گاه قوت بیشتری گرفته و گاه با ناباوری رنگ می بازد. با استناد به نظریات ماتریالیستها و نادیده انگاشتن واقعیات متافیزیکی و فراماده‌ای و چشم بستن بر معرفت درونی، انسان قرن بیست و یکمی، خیال خام بازتولید موجودی همانندِ خود را داشته و خوش‌خیالانه و بعضاً مذبوحانه در پیشبرد این مقصود جاه طلبانه می کوشد. این تلاشها گاه تا بدانجا پیش می رود که بشرِ عصرِ فن‌آوری، در صدد پیاده‌سازی احساسات و عواطف انسانی توسط فرمولهای ریاضی و الگوریتمهای پیشرفته برمی‌آید. خوشبختانه پس از طی سالیان و آزمودن روشها و نظریه های مختلف ریاضی، عالمان عالَم غالباً عاقبت بدین نتیجه‌ی تلخ رسیدند که اینگونه روشهای علمی کاملا دقیق که از افتخارات تاریخ علم بشریت محسوب می شد تا چه حد در شبیه سازی موجودات کیهان و کنترل فرایندهای طبیعی موجود ناتوانند.

با وجود ناکامی‌های فراوان، مقیمانِ عالمِ سفلی به عوض پرداختن به واقعیات فراماده و کشف روابط عرفانی و پذیرفتن جوهرِ وجودی انسان، پافشاری بر استخراجِ منطق از ساز و کارهای موجودِ کیهان را در پیش گرفت که یکی از نتایج آن به "تفکر فازی" معروف شد. با پیدایش این تلقی، روزنه امیدی در ذهن برخی اندیشمندان گشوده شد که شاید قادر باشند با دستاویز قرار دادن این نظریه‌ی نوظهور، رویاهای خود مبنی بر شبیه‌سازی موجودات عالم و بالاخص انسان را به منصه‌ی ظهور رسانند. دریغ که تا کنون کلیه ناکامیها و سرخوردگی‌ها را بر گردن سرعتِ پردازشِ پایین و یا عدمِ تکامل کافیِ ابزارهای فازی انداخته و ذره‌ای بدین نکته اندیشه نکرده اند که گلِ سرسبدِ مخلوقاتِ عالم، برخی قابلیتهای خود را نه با اتکا بر قوانین مادی و روشهای علمی که با اتصال به عالم غیرمادی به دست می‌آورد و با کوشش‌های مستمرِ علمی و مطالعاتِ تحقیقاتی نه می‌توان خوابهای محقق الوقوع تولید کرد و نه عشقی آنی تنها با یک تیرِ نگاهِ معشوق. با این وجود برخی قابلیتها را که انسان در مواجه با عالم مادی فرا می‌گیرد را می‌توان شبیه‌سازی و پیاده کرد و از آن جمله است طرز حرکت او.

تا کنون با استفاده از روشهای دقیق و یا فازی (به صورت استخراج خبرگی [1]) تلاشهای خوبی در زمینه ساخت روباتهایی که بتوانند همانند انسان قدم از قدم بردارند صورت گرفته است. در کوششهای پیشین سعی در راه بردن روبات با استفاده از روش استخراج خبرگی و یا *یادگیری فعال فازی*[i] بود که تا حدود قابل ملاحظه‌ای نیز موفقیت‌آمیز نمود. مبنای ALM بر این ایده استوار است که برای نوع بشر، به خاطر سپردن اعداد بسی مشکل‌تر از یادگیری رفتار است. به عبارت دیگر یادگیری یا مدلسازی انسان با توجه به سیستمهای ساده‌ی یک ورودی – یک خروجی (SISO) آغاز شده و سپس اثر سایر پارامترها بر آن مدل وارد می‌گردد و به این طریق درکی کلی و کیفی از رفتار سیستم به دست می‌آید[2]. در ادامه، کوشش بر این است که با استفاده از همان روش، دویدن روبات نه به صورت استخراج خبرگی که از طریق یادگیری خودکار از داده های موجود صورت گیرد.

روش یادگیری فعالی که در اینجا مطرح می شود، دارای دو رکنِ اساسی می باشد. یک رکن، تولیدِ قوانینِ فازی به صورت خودکار با استفاده از روش موثرترین ورودیست که در هر مرحله موثرترین ورودی را تشخیص

[i] Active Learning Method (ALM)

داده و به صورت درختی تا رسیدن به خطای مناسب ،ورودیهای دیگر را به شروط قوانین فازی می افزاید. رکن دیگر که شالوده کار را تشکیل می دهد، اصلاحِ بازه‌های توابعِ عضویت تولید شده (و نیز در شرایطی اصلاح قوانین فازی)، با استفاده از پس‌خورد موجود هنگام دویدن روبات است که می بایست به صورت خودکار صورت گرفته و اعمال شود. برای این مرحله روش تشخیص خطا و نیز چگونگی اصلاح توابع عضویت با استناد به خطای مشاهده شده اهمیت بسزایی دارد.

پس از طی مراحل فوق، به تدریج توابع عضویت و نیز قوانین فازی در جهت هرچه بهتر گام برداشتنِ روبات اصلاح شده و پس از طی زمانی چند، روبات قادر خواهد بود تا به روانی راه برود.

توضیح عملکرد

برای راه رفتن عادی، طبق نظر پرات پنج شرط باید برقرار باشد تا یک روبات دوپا و دوبعدی بتواند راه برود. ارتفاع، زاویه‌ی تعادل[ii]، و سرعت باید پایدار باشند، پای متحرک باید طوری حرکت کند که پاها در موقعیت مناسب برای حفظ ارتفاع، تعادل و سرعت قرار بگیرند و انتقال از حالت تکیه به یک پا به حالت تکیه به دو پا در زمان مناسب انجام گیرد[3]. برای عمل دویدن اما پایداری سرعت لزومی نمی یابد. چه ممکن است دویدن با شتاب ثابت و یا متغیر صورت پذیرد. لیکن پایداری ارتفاع و زاویه تعادل همچنان مطرح است که برای پسخورد سیستم و اصلاح داده ها استفاده خواهد شد.

یک راه برای شروع عملیات احتیاج به داده های تولید شده توسط یک روبات راه‌رونده می باشد.

راه دیگر پیش رو، تغییر آگاهانه قوانین و توابع عضویت فازی سیستم روبات راه رونده است. در بررسی داده های روبات راه رونده، در برخی شرایط مشاهده شد که هر دو پای روبات از زمین بلند می شوند. یکی از کارهایی که ابتدا می بایست انجام شود تامل در شرایطی است که موجبات این حرکت را فراهم می کنند. یعنی می بایست بررسی گردد که چه توابع عضویت و قوانینی باعث بلند شدن هر دو پای روبات از روی زمین می شوند. سپس شاید بتوان با استناد به تحلیل آن رفتار، توابع عضویت تولید شده برای راه رفتن معمولی روبات را به گونه ای تغییر داد که راه رفتن روبات را موجب شود.

در اینجا تنها به توضیح راه اول می پردازیم.

مانند قبل، معیار انتخاب مؤثرترین ورودی ، واریانس نقاط در صفحه‌ی آن نسبت به مسیر یافته شده است و اگر این پراکندگی از حد مشخصی فراتر رود، ورودی به عنوان یک ورودی غیر مؤثر تشخیص داده می‌شود و از لیست پارامترهای سیستم حذف می‌گردد.

در کار قبلی برای ساخت کنترل روبات راه رونده، دامنه‌ی متغیرها به بخش‌های کوچکتری تقسیم می شد و همین مراحل تا رسیدن به دقت کافی ادامه می‌یافت و سپس قوانین فازی به شکل زیر استخراج می‌شد.

then: Y is sU or tV if: X1 is A, and X2 is B, …

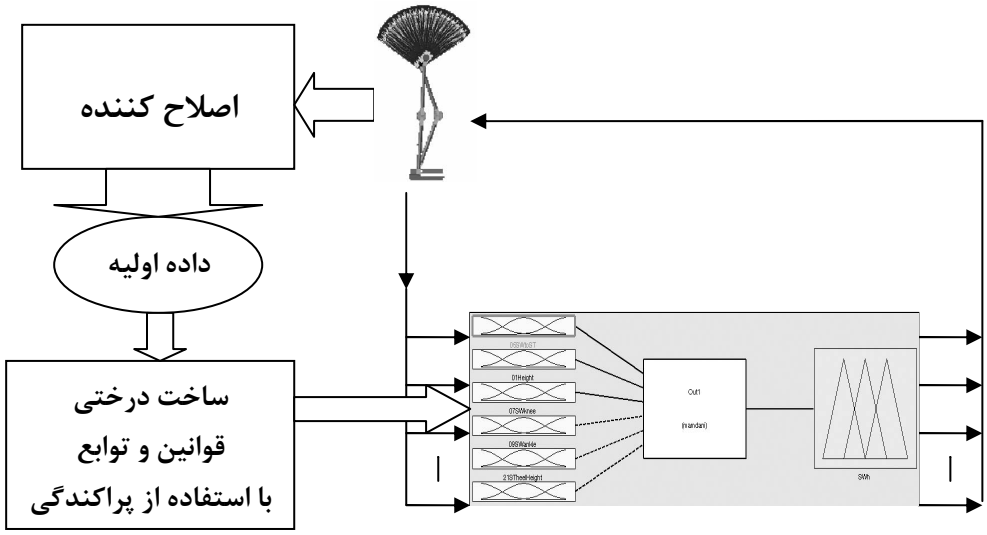
اینک اما روش دیگری به کار گرفته می شود.

در روش جدید، متغیرهای ورودی و خروجی توابع عضویت بسته به میزان پراکندگی، از همان ابتدا به سه و یا ۵ قسمت تقسیم می شوند و در هر زیر شاخه، متغیر اولیه حذف می شود. عملیات ساخت قوانین، مانند به

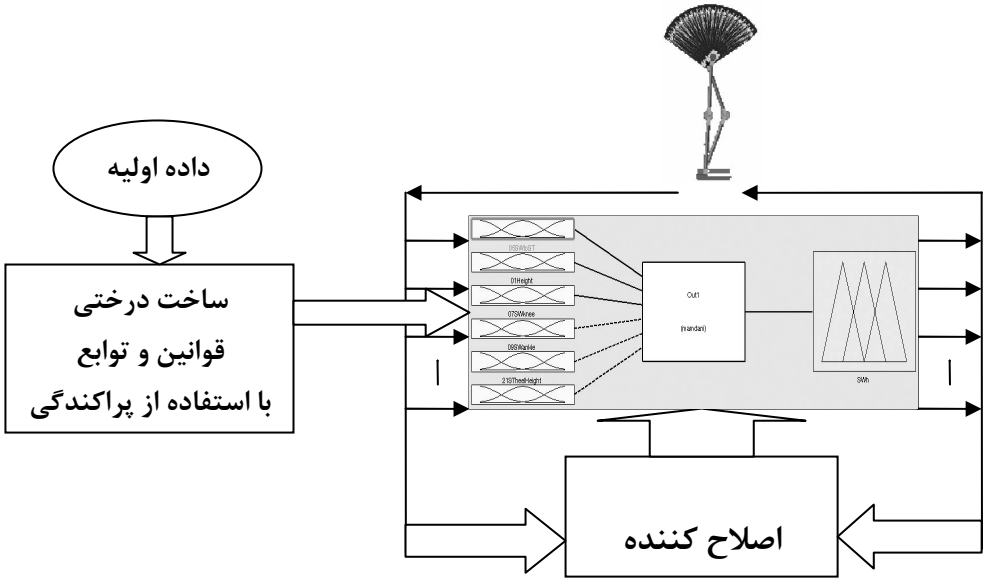
[ii] Pitch

صورت درختی تا رسیدن به خطای قابل قبول ادامه می یابد. در نهایت، مجموعه قوانین و توابع اولیه ما ساخته می شوند.

در مرحله بعد، می بایست روبات با استفاده از ساختار فازی موجود شروع به حرکت نماید. پس از مشاهده حرکت روبات در هر چرخه، (از زمان بلند شدن یک پا، تا زمان پایین آمدن پای دیگر) داده ها بررسی و توابع موجود با استفاده از میزان خطای آنها اصلاح می شوند. اصلاح کنترل کننده به دو صورت می تواند انجام شود. یک روش آن است که داده های اولیه اصلاح شده و دوباره راهکار فوق برای تولید ساختار کنترلگر فازی از ابتدا اعمال شود که در شکل زیر نمایش داده شده است.



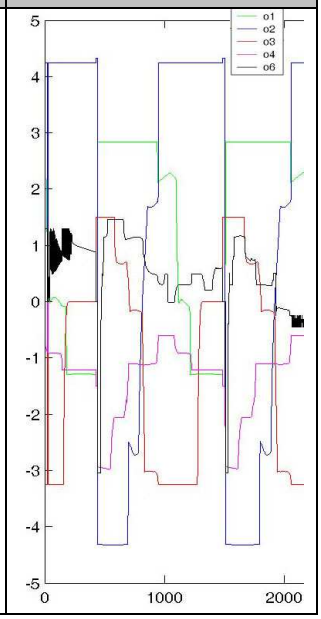
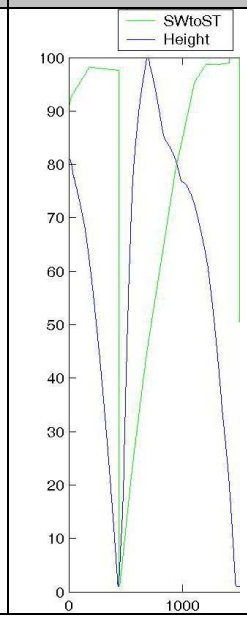
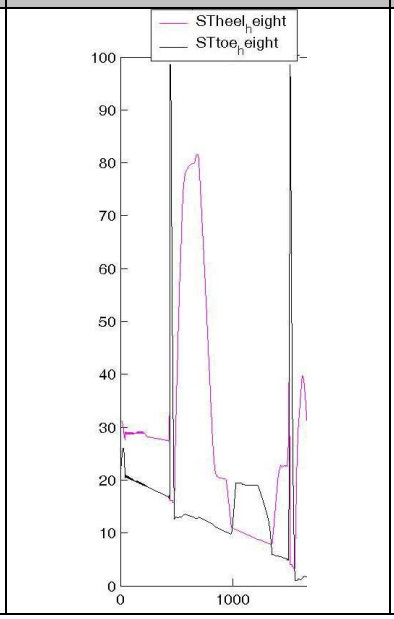
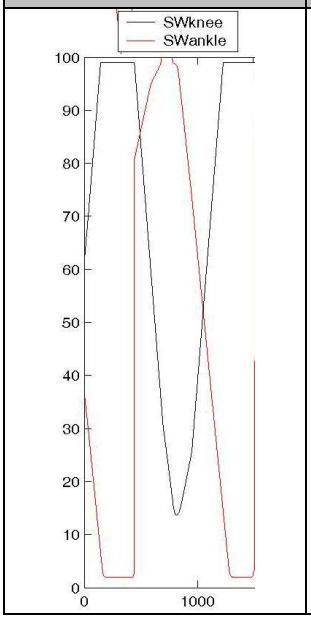
طریق دیگر اینست که تصحیح کننده، مطابق شکل، داده ها را جمع آوری کرده و ساختار کنترل کننده را مستقیما اصلاح کند. روش زیر چنانچه روش اصلاح روشی کارا باشد، بهتر و سریعتر جواب می دهد.



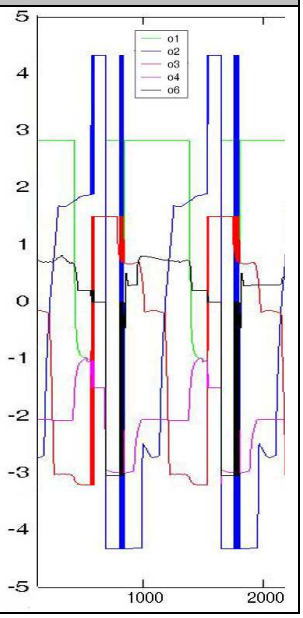
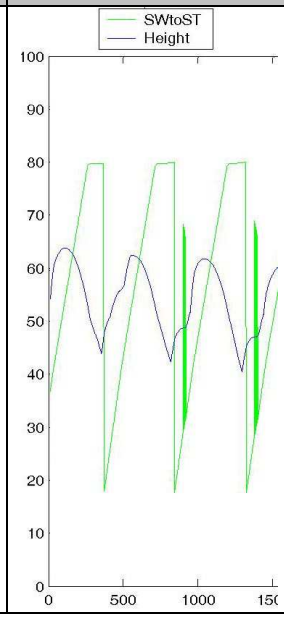
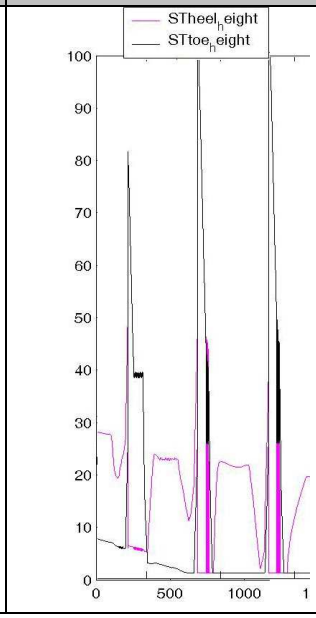
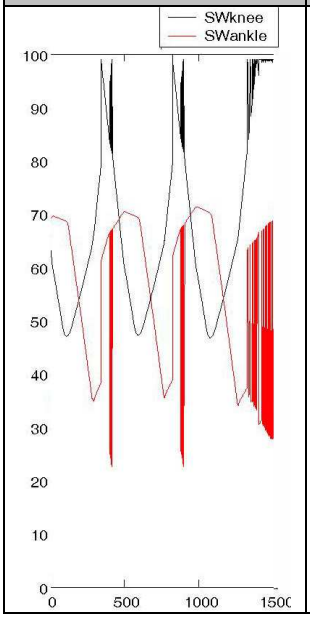
مشاهدات

هنگام کار بر روی روبات راه رونده، مشاهده شد چنانچه حداکثر ارتفاع روبات هنگام عمل نرمال سازی زیاد شود، شرط Height = high به ندرت برقرار می شود و این عمل باعث اجرا نشدن قانون مربوط به اعمال سرعت حرکت بالا به پای متحرک می شود. نتیجه آنکه سرعت حرکت پای متحرک یا حداکثر و یا حداقل می شود و این مسئله باعث نزدیک شدن پاهای ثابت و متغیر به یکدیگر می گردد. (SWtoSt کمتر شرط high را اکتساب می کند) چنین روندی باعث تندتر و نزدیکتر گام برداشتن روبات می گردد که نمودار ورودی و خروجی ها در شکل نشان داده شده اند:

در حالت راه رفتن عادی:

خروجیها	ارتفاع – فاصله پاها	ارتفاع پاشنه و انگشت پا(متحرک)	زاویه زانو و مچ پای متحرک
			

در حالت راه رفتن سریع و کوتاه:

خروجیها	ارتفاع – فاصله پاها	ارتفاع پاشنه و انگشت پا(متحرک)	زاویه زانو و مچ پای متحرک
			

همانطور که در داده های قبل مشاهده شد، چنانچه با تغییر قوانین سرعت حرکت پا و زانوی متحرک بیشتر و توابع عضویت آنها محدودتر گردد (سرعت تنها زیاد یا کم) روبات گامهای تند و کوتاه برخواهد داشت. این حرکت می تواند مقدمه ای بر دویدن آن باشد و به تدریج با پسخورد و مشاهده خطا اصلاح شود. همانگونه که یک کودک در ابتدای یادگیری دویدن، با سریع و ریز گام برداشتن شروع می‌کند و به تدریج با اصلاح حرکت خود یاد می‌گیرد چگونه به سهولت و بدون زمین خوردن راه برود.

مراجع

[1] Tabrizi S., Bagheri S., “Developing a fuzzy controller for a planar biped robot using human expertise extraction”, Proc. of International Symposium on Computational Intelligence and Intelligent Informatics (ISCIII'03), 2003

[2] Bagheri S., Honda N., “A new method for establishing and saving fuzzy membership functions”, Proc of the 13th Fuzzy Symposium, 1997

[3] Pratt J., Pratt G., “Intuitive Control of a Planar Bipedal Walking Robot”, Proc. Int. Conf. on Robotics and Automation (ICRA’98), 1998

فصل دوم: استخراج توابع و قوانین فازی با استفاده از منحنی های ورودی-خروجی

فهرست

تعاریف:

منحنی:

منحنی داده های یک ورودی بر حسب یک خروجی که از طریق روش IDS استخراج شده است.

خروجی:

خروجی در هر مرحله در ابتدای مرحله مشخص شده و تا انتهای کار ثابت می ماند.

شرح عملیات

هدف از عملیات، پیاده سازی و تولید یک ساختار مناسب خودکار برای تولید توابع عضویت و قوانین فازی کنترل کننده به روش ALM و با استفاده از IDS می باشد. برای این منظور، بازه های مناسب به طریقی که شرح داده خواهند شد استخراج و در هر مرحله بازه های خروجی با توجه به ورودی جدید شکسته می شوند و در نهایت با استفاده از بازه‌های موجود ورودی و خروجی قوانین فازی استخراج می شوند.

روش کار:

روش کلی کار بدین صورت است که در هر مرحله، ابتدا برای بازه موردنظر خروجی (که در مرحله اول کل بازه را در بر می‌گیرد) موثرترین ورودی را یافته و سپس توابع عضویت ورودی موثر مذکور و خروجی به وسیله پویش نقاط منحنی پیدا می شود. بازه خروجی جدید جایگزین بازه پیشین شده و ورودی جدید به همراه بازه‌اش در لیست قوانین ذخیره می شود. این عمل به صورت چرخه‌ای تا اتمام ورودیهای با پراکندگی مناسب در بازه بدست آمده تکرار می شود. (البته می توان عملیات را تا چند مرحله توافق شده نیز ادامه و سپس خاتمه داد) در پایان، از روی لیست داده بازه‌های موجود که عبارت از اطلاعات مربوط به بازه‌های ورودی-خروجی ، قوانین فازی استخراج و ذخیره می گردند.
شرح مراتب انجام عملیات هر مرحله به صورت زیر می باشد.

تعیین بازه مناسب

در ابتدا بازه موردنظر برای انجام عملیات تعیین می شود. در مرحله اول، بازه های هر دو مولفه ورودی و خروجی که از IDS گرفته می شوند،از ۱۰ تا ۱۱۰ حساب می شوند. زیرا از هر طرف هنگام IDS گیری به اندازه ۱۰ واحد گسترانیده شده اند.(بازه ای که پس از انجام IDS در اختیار داریم، ۰ تا ۱۲۰ می باشد). در مراحل بعد، بازه‌های مناسب از بازه‌های خروجی مرحله قبل به دست می‌آیند.

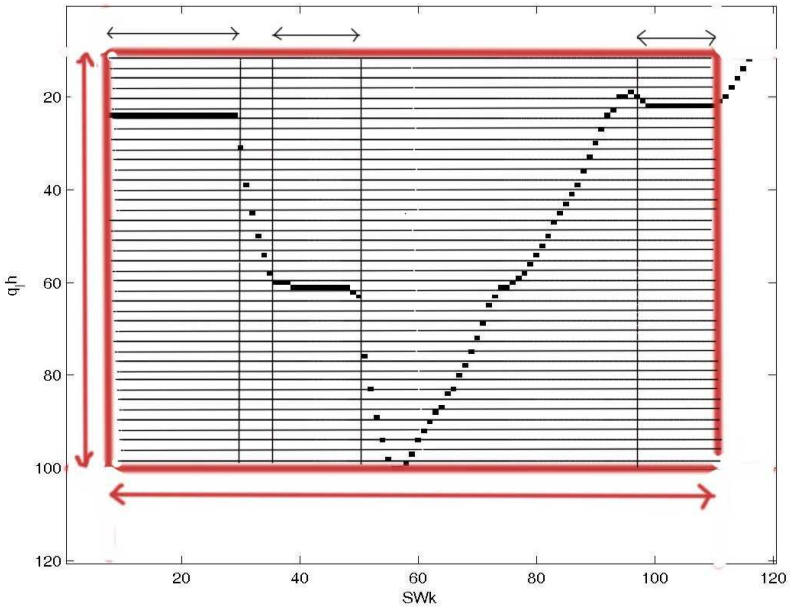
یافتن موثرترین ورودی

در اولین قدم پس از تعیین بازه، موثرترین ورودی در آن بازه برای این مرحله استخراج می شود. این کار به وسیله تعیین پراکندگی داده‌های ورودی‌ها و خروجی صورت می‌پذیرد. لازم به ذکر است که ورودیهای مراحل قبل از لیست ورودیها حذف می شوند. همچنین موثرترین ورودی، برای بازه موردنظر یافته می شود نه کل بازه خروجی. در مراحل بعد، پس از حذف داده‌های خارج از بازه به دست آمده از داده خام ورودی، بار دیگر IDS (این بار با سطح پراکندگی کمتر) برای ورودیها محاسبه و به وسیله پراکندگی بازه مورد نظر منحنی به دست آمده، موثرترین ورودی انتخاب می‌شود.

مشخص کردن بازه توابع ورودی خروجی

پس از مشخص شدن ورودی موثر، توابع ورودی و خروجی ساخته می شوند.(که البته توابع خروجی هر مرحله، ممکن است در مراحل بعد شکسته و کوچکتر شوند). طریقه ساخت این توابع بدین صورت است که بازه خروجی به قسمتهای مساوی تقسیم شده و برای هر شیار، تعداد نقاط پیوسته موجود در آن به همراه طول بازه نقاط مشخص می شود. (این عمل به صورت خط به خط انجام می گیرد) شاخص معتبر بودن هر شیار، تعداد نقاط داده موجود در آن شیار ضربدر طول بازه نقاط معتبر بعلاوه یک می باشد.

شکل:



شمارش نقاط یک بازه

نقاط یک بازه، برابر تعداد نقاط پیوسته هر خط می باشد. برای شمارش نقاط هر خط از شیار، اولین نقاط پیوسته موجود معیار قرار می گیرد. در خط بعد می‌بایست بازه نقاط به دست آمده در همسایگی مطلوبی از خط قبل باشد. چنانچه این شرط ارضاء نشود، بازه خط قوی‌تر (دارای نقاط بیشتر) برای تعیین بازه مناسب ملاک خواهد بود. جهت جلوگیری از حذف بازه های نقاط پیوسته ای که در یک شیار و یا حتی یک خط قرار گرفته اند، بدین صورت عمل می شود:

در هر شیار، چنانچه فاصله بازه نقاط هرخط با خط قبل از حد معینی فراتر رود، اگر خط جدید دارای نقاط بیشتر نسبت به خطوط قبلی باشد، شماره شیار در محلی ذخیره می شود و پس از اتمام عملیات، شیارهای به دست آمده دوباره بررسی می شوند. همچنین، به جهت از دست ندادن بازه های نقاط پیوسته‌ای که در یک خط قرار دارند، عملیات، بار دیگر از انتهای آخرین حد بازه معتبر به دست آمده در پایان کار تا انتهای بازه کلی تکرار می شود.

طریقه تعیین اعتبار

چنانچه مقدار عددی شاخص بدست آمده، بیشتر از حد مشخصی باشد (این حد به صورت تجربی به دست خواهد آمد و قابل تعیین به صورت پویا نیز می‌باشد که البته در اینجا به اندازه عرض یک شیار در نظر گرفته شده است) شیار به عنوان شیار معتبر برگزیده می شود. در درجه اعتبار آن نیز ذخیره می‌شود.

ذخیره مشخصات بازه ها

پس از عملیات فوق، مشخصات شیارهای معتبر در یک ماتریس ذخیره می شوند. این مشخصات عبارتند از : نقاط شروع و انتهای بازه های ورودی، نقطه شروع بازه خروجی، شاخص به دست آمده برای بازه و محل طولانی ترین خط پیوسته هر بازه.

مثلا دو سطر از ماتریس فوق می توانند بدین شکل باشند:

معتبرترین خط خروجی	شاخص اعتبار	ابتدای بازه خروجی	انتهای بازه ورودی	ابتدای بازه ورودی
11	961	10	35	5
16	9	15	38	36

از آنجایی که طول بازه های خروجی ثابت است، تنها نقطه شروع آنها برای داشتن مشخصاتشان کفایت می کند. معتبرترین خط خروجی در مراحل بعد برای تعیین بازه مناسب‌تر توابع عضویت خروجی به کار می رود.

شرط ترکیب شیارها:

در این مرحله ما آرایه‌ای دو بعدی از مشخصات شیارها داریم که در هر سطر آن نقطه شروع بازه خروجی، نقاط شروع و خاتمه‌ی بازه ورودی به همراه شاخص اعتبار و نیز معتبرترین خط خروجی آن شیار ذخیره شده اند. شیارها با توجه به مقدار شاخص اعتبار به دو نوع قوی و ضعیف تقسیم می شوند.

روش ترکیب بدین صورت است که هر شیار تنها در صورتی با شیار قبل از خود ترکیب می‌شود که دو شرط را حتما دارا باشد. ۱- بازه ورودی موجود در شیار قبل با بازه ورودی موجود آن فاصله زیادی نداشته باشد. ۲- بازه خروجی آن، همسایه بازه خروجی شیار قبل باشد. چنانچه این شروط برقرار باشند، دو شیار با یکدیگر تشکیل شیار واحد داده و مشخصات آن شیار واحد ذخیره می‌شود. این مشخصات به صورتی هستند که در مرحله بعد برای ساخت توابع عضویت

مورد استفاده قرار گیرند. برای هر یک از ورودی و خروجی ابتدا و انتهای بازه و نقطه (یا دو نقطه قوی) وسط بازه مشخص می‌گردند. نقاط وسط بازه بدین صورت تبیین می‌شوند:

اولین شیار:

خروجی شیار قوی:

ابتدا و انتها برابر ابتدا و انتهای بازه بعلاوه یک حاشیه می شود و وسط بازه برابر با قوی ترین خط آن شیار (در نقطه دوم) می شود.

خروجی شیار ضعیف:

ابتدا و انتها برابر ابتدا و انتهای بازه (بدون حاشیه)می شود و وسط بازه برابر با تقریبا وسط بازه می‌شود.

بازه ورودی: ابتدا و انتهای بازه آن شیار، به عنوان ابتدا و انتهای بازه، و وسط بازه شیار، در دومین خانه قرار می‌گیرد.

شیارهای بعدی:

چنانچه شیاری قوی باشد، برای خروجی نقطه قوی‌ترین خط و برای ورودی وسط بازه به عنوان نقطه وسط انتخاب می‌شود. چنانچه بازه پیشین نیز قوی بوده باشد، نقطه خط جدید در سومین خانه و در غیر این صورت در دومین خانه آرایه مشخصات قرار خواهد گرفت و خانه سوم صفر گذاشته می‌شود. بدین ترتیب اگر تنها یک بازه در مجموع بازه‌های ترکیب شده قوی باشد، یکی از خانه های دوم و یا سوم "صفر" گذاشته می‌شود و در صورتیکه بیش از یک بازه قوی وجود داشته باشد، خانه‌های دوم و سوم، اختصاص به نقاط قوی‌تر دو بازه قوی‌تر کل بازه‌های یک ترکیب خواهند داشت.

پوشش بازه های خالی خروجی

در اولین مرحله، بازه‌های خالی خروجی نیز پوشش داده می‌شوند. این بازه‌ها بدون ورودی به مرحله بعد فرستاده می‌شوند و در نتیجه، ورودی موثر به دست آمده برای این بازه در ابتدای مرحله بعد، اولین ورودی و ریشه درخت ورودیها خواهد بود.

ساخت توابع و قوانین فازی

ساخت توابع فازی از روی شیارهای متمایز به دست آمده از مرحله قبل، بدین شکل است.

در هر مرحله ساخت تابع با توجه به نوع شیار کمی متفاوت می شود.

توابع ورودی:

در مرحله اول، برای هر بازه خروجی، یک بازه ورودی وجود دارد. در مراحل بعد، برای بازه‌های خروجی موجود، موثرترین ورودی انتخاب و بازه خروجی برای آن ورودی جدید حساب می شود. حال، دو متغیر ورودی و یک بازه خروجی (بازه دوم) را در نظر گرفته و قانون به صورت "ورودی اول" و "ورودی دوم" آنگاه "خروجیِ ورودی دوم"

در می آید. چنانچه این عمل را برای خروجی دوم دوباره تکرار کنیم و تا n مرحله، ورودیهای موثر را بیابیم، قانون به صورت زیر در می آید.

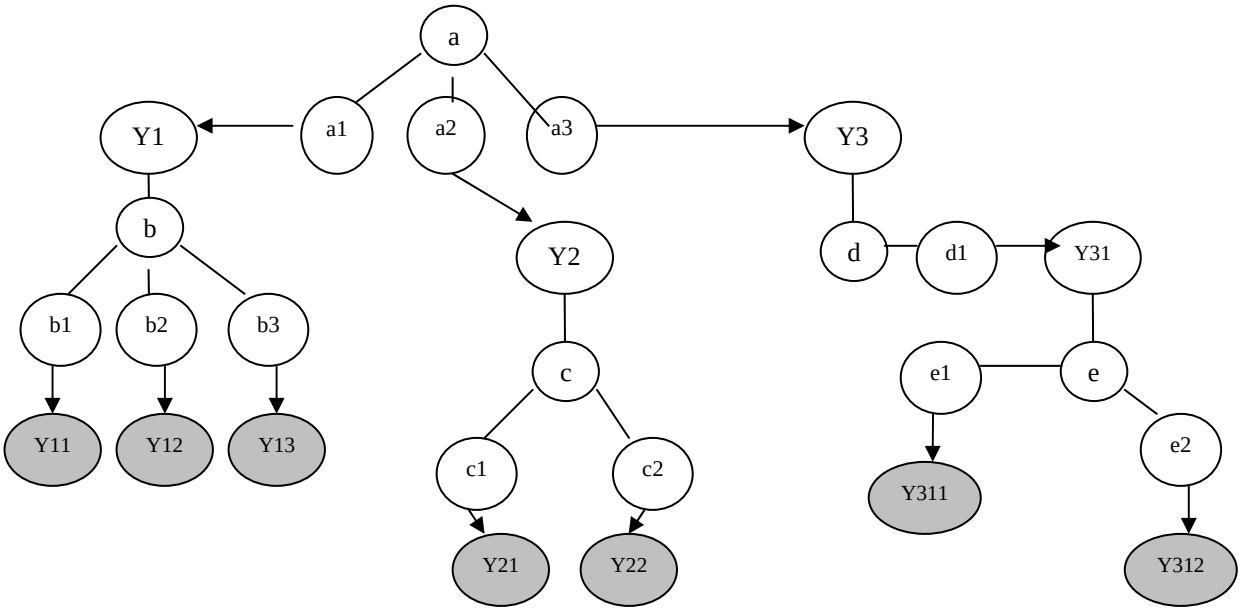
"ورودی اول" و "ورودی دوم" و .... و "ورودی n ام" آنگاه "خروجیِ ورودی n ام"

توابع خروجی:

مطابع آنچه درباره توابع ورودی ذکر شد، بازه خروجی، در انتهای چرخه به دست آورده بازه‌ها، برابر با کوچکترین بازه به دست آمده خواهد بود.

شکل درختی تولید:

چنانچه شکل درختی مراحل تولید قوانین برای خروجی y ، به صورت زیر می‌باشد،



قوانین بدین شکل در می‌آیند:

if (a is a1) and (b is b1) then y is y11
if (a is a1) and (b is b2) then y is y12
if (a is a1) and (b is b3) then y is y13
if (a is a2) and (c is c1) then y is y21
if (a is a2) and (c is c2) then y is y22
if (a is a3) and (d is d1) and (e is e1) then y is y311
if (a is a3) and (d is d1) and (e is e2) then y is y312

موارد آتی

برای ادامه کار می‌توان با استفاده از برنامه نوشته شده برای پیاده‌سازی موارد فوق، برخی از روشها را به سهولت تغییر داد. همچنین می‌توان دقت به دست آوردن شیارها و یا طریقه ترکیب آنها را به صورتی متفاوت انجام داده و نتایج را مشاهده نمود.

مورد قابل‌توجه دیگر اینکه محتمل است بدون استفاده از الگوریتم IDS که بسیار وقت‌گیر می‌باشد، تنها با استفاده از پراکندگی نقاط ورودی-خروجی، اقدام به استخراج بازه‌ها نمود که اگر بشود، چه دوغی خواهد شد.

فصلِ سوم: چگونگی پیاده‌سازی

فهرست

مقدمه

شرح کلی

هدف از این مستند، شرح چگونگی پیاده‌سازی ساخت کنترلگر فازی برای روبات Spring Flaming با استفاده از روش یادگیری فعال[iii] (ALM) به کمک IDS می‌باشد. مبنای ALM بر این ایده استوار است که یادگیری یا مدلسازی انسان با توجه به سیستمهای ساده‌ی یک ورودی – یک خروجی (SISO) آغاز می‌شود و سپس اثر سایر پارامترها بر آن وارد مدل می‌گردد و به این طریق درکی کلی و کیفی از رفتار سیستم به دست می‌آید. برای به دست آوردن قوانین حاکم بر سیستم یک ورودی-خروجی، از روش IDS برای استخراج مسیر داده استفاده می‌شود.

نرم‌افزارهای مورداستفاده

Yobotics

نرم افزار Yobotics برای پیاده‌سازی نرم‌افزاری کنترل روبات SpringFlamingo می‌باشد. با استفاده از این نرم‌افزار، متغیرهای وضعیت روبات داده و متغیرهای خروجی روبات گرفته می‌شوند و عملکرد آن با توجه به داده‌های اعمال شده قابل رؤیت می‌باشد.

JBuilder 9

نرم‌افزار JBuilder در حقیقت به عنوان واسطه بین Yobotics و Matlab عمل می‌کند. این نرم‌افزار داده‌های وضعیت روبات را از Yobotics اخذ و از طریق رابط Sucket به نرم‌افزار Matlab تحویل می‌دهد و مقادیر متغیرهای خروجی را نیز از مطلب دریافت کرده و به Yobotics جهت اعمال به روبات و نمایش حرکات تحویل می‌دهد.

Matlab 6.5

از این نرم‌افزار برای دو منظور استفاده می‌شود. ابتدا برای ساخت کنترلگر فازی با استفاده از داده‌های نمونه ورودیها-خروجیها ی روبات. و سپس برای محاسبه خروجی‌ها با استفاده از داده‌های وضعیت روبات و تحویل مقادیر خروجیها پس از محاسبه توسط کنترلگرهای فازی هریک از خروجیها.

روند ساخت کنترلگر فازی

کلیه مراحل مربوط به روند ساخت کنترگر، در فایلgenerate_fuzzy انجام می‌پذیرند. این تابع، تابع اصلی ساخت کنترلگر بوده و توسط آن، باقی توابع مربوط در جای خود صدا زده می‌شوند.

در هر مرحله، ابتدا برای بازه موردنظر خروجی (در مرحله اول کل بازه) موثرترین ورودی جستجو و سپس توابع عضویت ورودی موثر مذکور و خروجی به وسیله پویش نقاط منحنی پیدا می شود. بازه‌های خروجی جدید جایگزین بازه پیشین شده و ورودی جدید به همراه بازه‌اش با افزوده شدن به ورودی پیشین در لیست قوانین ذخیره می شود. این عمل به صورت چرخه‌ای تا اتمام ورودیهای با پراکندگی مناسب در بازه بدست آمده ویا تا چند مرحله مشخص ادامه داده می‌شود. در پایان، از روی لیست داده موجود که عبارت

[iii] Active Learning Method

از بازه‌های ورودی-خروجی به همراه شاخص ورودی و شماره آن و نیز شاخص خروجی می باشد، قوانین فازی استخراج و در یک فایل fis ذخیره می گردند.

استخراج بازه‌ها

بازه‌ها در حقیقت عبارتند از نقاط شروع، میان و پایان توابع عضویت برای ورودیها و خروجی که از روی نمودار منحنی به دست می‌آیند. هر بازه ورودی دارای شماره ورودی و نیز شاخص ورودی (شماره تابع عضویت آن ورودی) می‌باشد. بازه‌های خروجی تنها دارای شماره تابع می‌باشند. بدین طریق در پایان استخراج بازه‌ها، آرایه دوبعدی‌ای خواهیم داشت که هر سطر آن معرف یک بازه ورودی و خروجی به همراه اطلاعات جانبی می‌باشد. لازم به ذکر است بازه ورودیهایی که در بیش از یک قانون استفاده می‌شوند، تکرار می‌شود. همچنین شماره تابع خروجی، بیانگر شماره قانون نیز می‌باشد زیرا برای هر قانون، تنها یک بازه خروجی در نظر گرفته می‌شود. اطلاعات یک سطر در پایان عملیات استخراج مشابه زیر خواهد شد:

۲۰	.	۴۵	۶۰	۱۰	۱۵	.	۲۰	۱	۱	۱
۶۵	۷۰	.	۱۰۰	۴۰	۴۵	۵۵	۶۰	۲	۱	۲

ستونهای ۱ تا ۴ متعلق به بازه‌ی ورودی، ۵ تا ۷ بازه خروجی، ۹ شاخص ورودی، ۱۰ شماره ورودی و ۱۱ شماره خروجی می‌باشند.

برای به دست آوردن ماتریس فوق‌الذکر، مراحل زیر به ترتیب می‌شوند.

اسختراج ورودیهای موثر

مؤثرترین ورودی، پارامتری است که قادر است باریکترین مسیر پیوسته را تولید کند یا به عبارت دیگر داده‌های خروجی بر حسب آن کمترین پراکندگی را داشته باشند. ورودی موثر در هر مرحله، برای یک بازه معین از خروجی جستجو می‌شود. این بازه در ابتدا برابر با کل بازه بوده و در مراحل بعد از بازه‌های خروجی به دست آمده از مرحله پیشین (و توسط تابع getValidOutputDomains) به دست می‌آید. پس از تعیین بازه خروجی موردنظر، تابع getMostEfficientInputSpec صدا زده شده و توسط آن موثرترین ورودی (به همراه میزان پراکندگی و نیز ماتریس نمودار آن بر حسب خروجی موردنظر) استخراج می‌شود. در هر مرحله جدید، ورودیهای مرحله قبل از لیست ورودیها حذف می‌شوند.

تشخیص شیارهای مناسب

پس از مشخص شدن موثرترین ورودی، شیارهای مناسب استخراج می‌شوند. شیارهای مناسب به شیارهایی اطلاق می‌گردد که از تعداد قابل قبولی نقطه (متناسب با بازه ورودی) در نمودار ورودی-خروجی برخوردار باشند. استخراج شیارهای مناسب توسط تابع getValidDomains انجام می‌پذیرد. مشخصات شیارهای یافت شده توسط این تابع در

یک آرایه دوبعدی برگردانده می‌شود که هر سطر آن به ترتیب شامل نقاط شروع و انتهای بازه ورودی، نقطه شروع بازه خروجی، شاخص اعتبار شیار و نیز محل قوی‌ترین خط شیار خروجی می‌باشد.

۴۵	۶۵	۷۰	۴۴۱	۷۱
۸۱	۹۸	۸۵	۲۷۰	۸۶

ترکیب شیارها

شیارهایی که در آرایه‌ی دوبعدی توسط تابع getValidDomains به دست آمده‌اند، به تابع getMergedDomainList فرستاده می‌شوند تا شیارهای مناسب با یکدیگر ترکیب شده و تشکیل یک بازه بدهند. آرایه برگردانده شده توسط این تابع دارای مشخصات آرایه نهایی می‌باشند. عناصر این آرایه، پس از اصلاح سطور آرایه کلی، به سطور آن اضافه می‌شوند. این اصلاح عبارت از اصلاح بازه خروجی و مطابقت آن با بازه خروجی کنونی می‌باشد. زیرا مطابق با الگوریتم، خروجی نهایی برابر با کوچکترین خروجی به دست آمده از مجموع ورودیها می‌باشد. (ر.ک. استخراج قوانین و توابع ...)

بازگشت به ابتدای چرخه

پس از طی مراحل فوق، دوباره از ابتدا برای خروجی‌های به دست آمده، ورودیهای موثر (مرحله اول) جستجو شده و این چرخه تا رسیدن به دقت مطلوب ادامه می‌یابد. این دقت می‌تواند از تعداد شاخه‌های درخت تولید قانون و یا از میزان پراکندگی ورودیها و یا حتی از اعمال به روبات و مشاهده نتایج (البته پس از استخراج توابع و قوانین) گرفته شود.

استخراج توابع و قوانین

استخراج توابع و قوانین از روی (نرمال شده‌ی) ماتریس نهایی به دست آمده از مرحله‌ی قبل صورت می‌پذیرد. در ابتدا توابع استخراج و سپس قوانین تولید می‌شوند. این قسمتها توسط تابع createFuzzyRules انجام می‌پذیرد.

توابع

برای به دست آوردن توابع، ابتدا آرایه بر حسب شماره ورودیها مرتب شده و سپس برای هر ورودی با استفاده از شاخص آن و بازه‌های به دست آمده تابع عضویت به دست می‌آید. چنانچه یکی از نقاط ۲ یا ۳ بازه‌ی موردنظر صفر باشد، تابع مثلثی و در غیر این صورت ذوزنقه‌ای خواهد بود. برای خروجی‌ها نیز به همین ترتیب عمل می‌شود با این تفاوت که خروجیهای تکراری حذف می‌گردند. اسامی توابع ورودی، به صورت شماره آن ورودی در ماتریس داده‌های اولیه به اضافه نام آن ستون انتخاب می‌شود تا هنگام انجام عمل کنترل، بتوان با استفاده از شماره آن ورودی، داده موردنظر را از داده‌های فرستاده شده تمیز داده و عمل کنترل را انجام داد.

قوانین

برای به دست آوردن قوانین، تنها سه عنصر آخر آرایه‌ی موردنظر کفایت می‌کند. ابتدا داده‌ها بر حسب شماره خروجی مرتب شده و سپس آرایه قوانین بر حسب شماره و شاخص ورودیها ساخته می‌شود. اندازه تعداد ستونهای آرایه قوانین برابر تعداد ورودیها به علاوه ۳ می‌باشد. زیرا یک خروجی بیشتر نداشته و دو عنصر آخر نیز همواره ۱ هستند. البته عنصر یکی مانده به آخر آرایه وزن قانون را نشان می‌دهد که می‌تواند بر حسب قدرت بازه از ۰ تا ۱ تغییر کند. مثلا برای آرایه‌ای با مشخصات زیر :

شماره خروجی	شماره ورودی	شاخص ورودی
۱	۱	۱
۲	۲	۱
۳	۲	۲
۱	۳	۱
۳	۴	۱

آرایه‌ی قوانین به صورت زیر در خواهد آمد:

۱	۱	۱	۰	۱	۰	۱
۱	۱	۲	۰	۰	۱	۰
۱	۱	۳	۱	۰	۲	۰

چگونگی اعمال کنترلگر به روبات

برای کنترل روبات، داده‌های هر لحظه توسط کلاس DanielOlivavFuzzyController جمع‌آوری و در Socket(3333) ریخته می‌شوند. داده‌های این سوکت توسط تابع daniel_must_do خوانده و قوانین فازی بر آنها اعمال و خروجیها مشخص می‌گردند. مقادیر خروجیها پس از محاسبه در سوکت ذخیره و توسط همان کلاس جاوا به روبات اعمال می‌شوند. داده‌ها به همان ترتیب که در ماتریس داده اولیه ورودیها-خروجیهای سیستم ذخیره شده‌اند فرستاده می‌شوند.

کسب داده‌های وضعیت روبات

داده‌ها توسط کلاس DanielOlivavFuzzyController فرستاده و توسط daniel_must_do خوانده می‌شوند. داده‌های مربوط به هر خروجی توسط تابع getInputs خوانده می‌شوند. این تابع از روی قوانین فازی مربوط به هر خروجی، شماره ورودیهای موردنیاز را استخراج کرده و داده‌ی مربوط به آنها را در یک آرایه برمی‌گرداند.

اعمال داده‌ها به کنترلگر

مقادیر داده‌های به دست آمده از تاعب getInputs سپس با استفاده از تابع evalfis به کنترلگر فازی مربوط به هر خروجی اعمال و مقادیر به دست آمده برای خروجیها پس از انجام عمل denormalize در یک ماتریس ذخیره می‌شوند.

فرستادن مقادیر خروجی به روبات

پس از به دست آمده مقادیر خروجیها، این مقادیر دوباره به سوکت فرستاده می‌شوند تا پس از خوانده شدن توسط کلاس DanielOlivavFuzzyController به روبات اعمال شوند.

فصل چهارم: شرحِ توابع اساسی برنامه‌ی تولیدکننده‌ی کنترل‌کننده

فهرست

تابع createFuzzyRules(outFile,mfList,dataList)

شرح عملکرد

متغیرهای ورودی

outFile : نام فایل موردنظر برای ذخیره قوانین و توابع (fis.)

mfList: آرایه حاوی مشخصات بازه‌های ورودی و خروجی و شاخص و شماره‌ی آنها. شماره‌ی خروجی به عنوان شماره‌ی قانون نیز می‌تواند مورد توجه قرار گیرد، زیرا هر خروجی موجود بیانگر یک قانون است.

dataList: آرایه حاوی اسامی ورودیها که شماره هر سطر، شماره‌ی ورودی موجود در آرایه‌ی mfList می‌باشد. (ستون ۱۰)

شرح کد

```
function y=createFuzzyRules(outFile,mfList,dataList);
a=newfis('pujan');
```

برای ساخت فایل مشخصات فازی، ابتدا توابع فازی مشخص و سپس در مرحله بعد، قوانین از روی آرایه مشخصات داده شده (mfList) استخراج می‌شود.

برای استخراج توابع، ابتدا آرایه‌ی داده شده بر حسب شماره‌ی ورودی و شاخص هر ورودی مرتب می‌شود.

```
mfList=sortrows(mfList,[10,9]) %sort by inputNumber
```

شماره‌ی اولین ورودی (معمولا "۱") در متغیر زیر ذخیره می‌شود.

```
inputNum = mfList(1,10)%first row
```

نام ورودی از ستون اول و نام خروجی از ستون دوم سطر برابر شماره‌ی ورودی آرایه‌ی dataList خوانده می‌شود.

```
inputName=dataList(inputNum,1);
outputName = dataList(inputNum,2);
```

سپس نام و بازه‌ی متغیرهای ورودی و خروجی توسط تابع addvar به مشخصه‌ی فازی تعریف شده (a) افزوده می‌شود.

```
a=addvar(a,'input',char(inputName),[0 100]);%first Variable
a=addvar(a,'output',char(outputName),[-100 100]);
```

شماره‌ی توابع هر ورودی، در متغیر زیر ذخیره می‌شود.

```
inputMF_Index=1;
```

شماره‌ی تابع خروجی، در متغیر زیر ذخیره می‌شود.

```
outputMF_Index=1;
```

متغیر زیر برای تشخیص ورودی جدید به کار می‌رود. بدین صورت که در حلقه تولید توابع، شماره‌ی آخرین ورودی در آن ذخیره می‌شود و چنانچه تغییر کند، یعنی به ورودی جدید رسیده‌ایم.

```
lastInputNumber=0;
```

از آنجائیکه آرایه مشخصات بر حسب شماره ورودیها و شاخص آنها مرتب شده است، شماره‌ی تابع خروجی در آن به صورت پراکنده است. برای آنکه از اختیار تابع خروجی تکراری هنگام استخراج توابع اجتناب شود، هر بار خروجی گرفته شده را در آرایه‌ی زیر ذخیره می کنیم و در چرخه‌ی بعد، محتویات این آرایه را بررسی می‌کنیم و چنانچه شماره‌ی خروجی موجود در آن نبود خروجی جدید بوده و برای آن تابع اختصاص می‌دهیم.

```
outs=[];
```

هنگام ورودی جدید این مقدار یک و در غیر این صورت ۰ می‌باشد.

```
new=1;
mfListSize=size(mfList)
```

در حلقه‌ی زیر به ترتیب سطور آرایه‌ی mfList مورد بررسی قرار می‌گیرند و توابع ورودیها و خروجی استخراج می‌شوند.

```
for(i=1:mfListSize(1))
```

شرط زیر هرگاه برقرار باشد، یعنی به ورودی جدید رسیده‌ایم که می‌بایست به سیستم فازی افزوده گردد (addvar)

```
  if(inputNum ~= mfList(i,10))
    disp('new Input');
    %adding new input variable
    inputName = dataList(mfList(i,10),1);
    a=addvar(a,'input',char(inputName),[0 100]);
    inputNum = mfList(i,10);
```

شماره‌ی تابع ورودی جدید طبیعتا برابر یک می‌شود.

```
    inputMF_Index=1;
    new=1;
  end


  if(lastInputNumber~=mfList(i,9)||new==1)
    %removing -0- if exists in domain
    new=0;
```

از آنجا که بعضی بازه‌ها سه مقداره (یک "۰" در عنصر دوم و یا سوم) هستند، با استفاده از تابع زیر، آرایه‌ی حاوی مقادیر صحیح برگردانده می‌شود که سه یا چهار عنصری است.

```
  validInDomain=getValidDomain([mfList(i,1),mfList(i,2),mfList(i,3),mfList(i,4)]);
    domSize=size(validInDomain);
```

در صورتیکه متغیر validInDomain ۳ عنصر داشته باشد، تابع ما مثلثی و درغیر این صورت ذوزنقه‌ای است.

```
    if(domSize(2)==3)
      type='trimf';
    else
      type='trapmf';
    end
```

پس از مشخص شدن نوع تابع، به ورودی مورد نظر افزوده می‌شود.

```
    a=addmf(a,'input',inputNum,['in',num2str(inputMF_Index)],type,validInDomain);
```

شماره‌ی تابع ورودی کنونی یکی افزوده می‌گردد.

```
    inputMF_Index=inputMF_Index+1;
    lastInputNumber=mfList(i,9);
  end
```

در قسمت زیر، تابع خروجی در صورتیکه در چرخه‌های پیشین مشخص نشده باشد (در outs نباشد) مشخص می‌گردد. باقی مراحل شبیه افزودن تابع ورودی می‌باشد.

```
  if(isempty(find(outs==mfList(i,11))))
    validOutDomain=getValidDomain([mfList(i,5),mfList(i,6),mfList(i,7),mfList(i,8)]);
    domSize=size(validOutDomain);
    if(domSize(2)==3)
      type='trimf';
    else
      type='trapmf';
    end
    a=addmf(a,'output',1,['out',num2str(mfList(i,11))],type,validOutDomain);
    %add this ouputIndex to the array
    outsSize=size(outs);
```

پس از افزودن تابع شماره‌ی آن در outs ذخیره می‌شود تا در چرخه‌های بعدی دوباره افزوده نگردد.

```
        outs(outsSize(2)+1)=mfList(i,11);
      end%if output not tekraari
   end%for
```

ساخت توابع فازی در بالا پایان می‌یابد. در قسمت زیر، ماتریس مربوط به قوانین ساخته می‌شود.

```
 %=================NOW EXTRACTING FUZZY
RULES=====================

%  inputInex  inputNumber outputIndex
% [2         1          3           ]
 inoutList = [mfList(:,9),mfList(:,10),mfList(:,11)];
 inoutList = sortrows(inoutList,3);%sort by outputIndexNumbers ( ruleNumbers)
 inoutSize = size(inoutList);
lastRule = inoutList(1,3);
ruleNum = 1;
Number_of_inputs = getNumberOfInputs(inoutList(:,2))
for(i=1:inoutSize(1))
   if(lastRule == inoutList(i,3))
       ruleList(ruleNum,inoutList(i,2))=inoutList(i,1);
       ruleList(ruleNum,Number_of_inputs+1)=inoutList(i,3);
    else % = new rule
       ruleNum = ruleNum+1;
       ruleList(ruleNum,inoutList(i,2))=inoutList(i,1);
       ruleList(ruleNum,Number_of_inputs+1)=inoutList(i,3);
      lastRule = inoutList(i,3);
    end

 end
ruleList(:,Number_of_inputs+2)=1;
ruleList(:,Number_of_inputs+3)=1;
 showRule = ruleList
%ruleList=[
%1 0 0 1 1 1
%2 0 2 3 1 1
%1 1 1 2 1 1];
a=addrule(a,ruleList);

writefis(a,outFile);
```

تابع generate_fuzzy(myData,colName)

شرح عملکرد

این تابع در حقیقت اصلی‌ترین تابع برای ساخت کنترلگر فازی است که باقی توابع موردنیاز توسط آن صدا زده می‌شوند. با صدا زدن این تابع، از روی داده‌های فرستاده شده به آن که در حقیقت داده‌های وضعیتی یک روبات هستند، توابع و قوانین موجود برای هر خروجی ساخته می‌شود.

متغیرهای ورودی

myData : ماتریسی از داده‌های ورودی و خروجیهای روبات راه‌رونده می‌باشد. این داده معمولا از workspace خود Matlab گرفته می‌شود که طریقه ساخت آن نیز معمولا از روی فایل حاوی داده‌ها (به صورت csv) صورت می‌گیرد.

colName: آرایه حاوی نام ستونها به ترتیب موجود در فایل myData می‌باشد که برای استخراج نام متغیرها مورد استفاده قرار می‌گیرد. نرم‌افزار matlab هنگام بار کردن فایل حاوی داده‌ها، سطر اول آن که حاوی نام ستونها می‌باشد را در متغیری مجزا در workspace ذخیره می‌کند.

شرح کد

```
function y=generate_fuzzy(myData,colName);

outputNumber = 6;
```

در این قسمت داده‌های گرفته شده از ۰ تا ۱۰۰ نرمال می‌شوند.

```
qData=IDS_quantize(myData);
cols = size(qData(1,:));
rowSize=cols(2)-outputNumber;
outputColumnBeginIndex = cols(2)-outputNumber+1
```

در متغیرهای زیر، نقاط شروع و پایان برازش منحنی ورودی-خروجی تعیین می‌گردد. از آنجا که خروجی حاصل از IDS بین ۰ تا ۱۲۰ می باشد، و از هر طرف ۱۰ واحد اضافه فرض شده است، نقاط شروع و پایان برای ورودی و خروجی فعلا ۵ تا ۱۱۵ در نظر گرفته می‌شود.

```
inputBeginIndex = 5;
outputBeginIndex = 5%important! if new domain is from 5 to 60 :115-60;
inputEndIndex = 115;
outputEndIndex = 115%important! if new domain is from 5 to 60 :115-5;
```

متغیر زیر طول شیارهای خروجی هر مرحله را تعیین می‌کند. هرچه طول شیارها کمتر باشد، دقت استخراج منحنی بیشتر خواهد بود.

```
BandLength = 5;
```

متغیر زیر حد بالایی دامنه توابع عضویت ورودی را نشان می‌دهد.

```
mfDomainLimit=100;
```

عملیات ساخت قوانین برای هر خروجی به طور مجزا تکرار می‌شود، نتیجتا برای پوشش کلیه خروجی‌ها از یک حلقه استفاده می‌کنیم و اندیس خروجی در متغیر j مورداستفاده قرار می‌گیرد.

```
for (j=outputColumnBeginIndex:cols(2));%all outputs
```

در ستون اول متغیر زیر، نام ورودیهای به دست آمده و در ستون دوم، نام خروجی ذخیره می‌شود. شماره هر سطر، برابر با شماره ترتیب ورودی به‌دست آمده می‌باشد.

```
dataNameList=cell(1,2);
```

متغیر زیر برای ذخیره مشخصات بازه‌ها و نیز مشخصه‌های ورودی و خروجی (شماره ورودی، خروجی و شاخص ورودی) می‌باشد. در اینجا در ابتدای کار تهی انتخاب می‌شود.

```
pureMergedList=[];
```

متغیرهای زیر، نقاط شروع و انتهای بازه خروجی را برای عملیات استخراج توابع مشخص می‌کنند که در ابتدای کار، کل بازه یعنی ۰ تا ۱۰۰ را در بر می‌گیرد.

```
startDomain=0;
endDomain=100;
```

شرط زیر بدین علت است که خروجی پنجم (ستون ۳۱ ماتریس داده‌ها) ثابت می‌باشد و به محاسبه نیازی نیست، پس به سراغ خروجی بعدی می‌رود.

```
if(j==31)
   continue;
end
```

نام فایل موردنظر برای ذخیره مشخصات فازی کنترلگر این خروجی را مشخص می‌کند.

```
ouputFileName=char(strcat('input26_fuzzyOut_',colName(1,j)))
```

قسمت زیر مختص به پیدا کردن اولین ورودی موثر و یافتن بازه‌های آن می‌باشد که برای راحتی از قسمتهای بعدی جدا در نظر گرفته شده است.

در هر مرحله، متغیرهایی که یافته می‌شوند از لیست متغیرهای حذف می‌گردند. این متغیرها در آرایه‌ی زیر ذخیره می‌گردند.

```
usedInputs=[];
```

تابع زیر موثرترین ورودی را در بازه داده شده (که در اینجا کل بازه یعنی از ۵ تا ۱۱۵ است) می‌یابد و شماره‌ی آن (i) را به همراه نمودار IDS تولید شده (q) و نمودار پخش جوهر(s) و نیز معیار پراکندگی برمی‌گرداند.

```
[i,q,s,dest] =
getMostEfficientInputSpec(i_f,qData,usedInputs,j,colName,startDomain,endDomain);
```

شماره ورودی در متغیر زیر ذخیره می‌شود. شماره‌ی ورودیها صعودی و از ″۱″ شروع می‌شوند.

```
inputNumber=1;
```

با استفاده از متد getValidDomains مشخصات بازه‌های ورودی موثر در بازه خروجی داده شده برگردانده می‌شود. این مشخصات پس از مرتب شدن بر حسب بازه خروجی، در متغیر زیر ذخیره می‌گردند.

```
domains=sortrows(getValidDomains(q,outputBeginIndex,outputEndIndex,inputBeginIndex,in
putEndIndex,BandLength),3)
```

متد getMergedList مشخصات بازه‌های گرفته شده فوق را دریافت و بازه‌های مناسب را با یکدیگر ترکیب می‌نماید. سپس آرایه خروجی در متغیر زیر ذخیره می‌شود. این متغیر همان متغیر نگهدارنده مشخصات توابع و قوانین است که در انتها به تابع createFuzzyRules فرستاده می‌شود تا فایل fis موردنظر ساخته شود.

```
pureMergedList=getMergedDomainList(domains,inputNumber,1,5,5)% bandlength maybe 2
```

شماره ورودی در ماتریس داده، به همراه اسم آن در متغیر dataNameList ذخیره می‌شود تا هنگام ساخت توابع و قوانین فازی از آن استفاده شود.دو شناسه‌ی اول به عدد شماره اختصاص دارد و چنانچه عدد یک رقمی باشد، شناسه‌ی اول ″۰″ گذاشته می‌شود.

```
if(i<10)
   varNum=['0',num2str(i),'_',char(colName(1,i))];
else
   varNum=[num2str(i),'_',char(colName(1,i))]
```

```
end
columnName=cellstr(varNum);
dataNameList(inputNumber,:)=[columnName,colName(1,j)];
```
ورودی یافته شده به متغیر زیر افزوده می‌شود.
```
usedInputs(1)=i;
```
==========

در این قسمت شاخه‌های زیرین پویش می‌شوند.

مطابق این حلقه، عملیات تا حداکثر ۳ شاخه ادامه پیدا می‌کند. (که می‌تواند بیشتر نیز باشد)

```
for(k=2:3)
   i_f=0;

if(k==2) % to cover empty space in the first level
   covEmpSp=1;
else
   covEmpSp=0;
end
outDomains=getValidOutputDomains(pureMergedList,covEmpSp)
   outDomainsSize=size(outDomains);
   outDomLength = length(outDomains);
   newInputNumberIndex=1; %the index of new input
   thisLevelUsedInputs=[];
   for(d=1:outDomainsSize(1))
      startDomain=100/120*(115-outDomains(d,4))
      endDomain=100/120*(115-outDomains(d,1))
      %getting the most efficient input in the given domain
      [i,q,s,dest] =
getMostEfficientInputSpec(i_f,qData,usedInputs,j,colName,startDomain,endDomain);
      if(isempty(q))
        continue;
      end
      paraakandegi=dest
   %-----------------------------------------------------------------


      thisLevelOutputBeginDomain=round(115-outDomains(d,4))
      thisLevelOutputEndDomain=round(115-outDomains(d,1))

returnedDomains=getValidDomains(q,thisLevelOutputBeginDomain,thisLevelOutputEndDo
main,inputBeginIndex,inputEndIndex,BandLength)
      if(isempty(returnedDomains)~=1)
         if(returnedDomains(1)<10 && returnedDomains(2)>=110) % output constant for all
range of input
            continue;
         end
         if(isempty(find(thisLevelUsedInputs==i))==1)%this input not exists
            thisLevelUsedInputs(numel(thisLevelUsedInputs)+1)=i %for holding the input
numbers to be avoided at the next levels.
            inputNumber=inputNumber+1;%can be different from k
            if(i<10)
               varNum=['0',num2str(i),'_',char(colName(1,i))];
```

```
            else
               varNum=[num2str(i),'_',char(colName(1,i))];
            end
            columnName=cellstr(varNum)
            dataNameList(inputNumber,:)=[columnName,colName(1,j)]
            newInputNumberIndex=1; %the index of new input
          else % input tekraari
            inputNumber=find(thisLevelUsedInputs==i)+length(usedInputs)
            newInputNumberIndex=getMaxNumberIndex(pureMergedList,inputNumber)+1;
%the index of the old input====================ERROR!=========
          end
          domains=sortrows(returnedDomains,3)

pureMergedList1=getMergedDomainList(domains,inputNumber,newInputNumberIndex,5,5,o
utDomains(d,5)*10*k)
          %adding rows to previous rows and clowning the previous rows by the number
          %of new rows (for extracting rules)
          p1size=size(pureMergedList1);
          b=0;
           for(a=1:p1size(1))
             newInputNumberIndex=newInputNumberIndex+1;

newOutDomain=[pureMergedList1(a,5),pureMergedList1(a,6),pureMergedList1(a,7),pureMe
rgedList1(a,8),pureMergedList1(a,11)]
             pmsize=size(pureMergedList);
            if(a==1) % first time, manipulating
               outRow=0;
               for(b=1:pmsize(1))
                  if (pureMergedList(b,11)==outDomains(d,5))
                  % manipulating previous row
                    pureMergedList(b,5)= newOutDomain(1,1);
                    pureMergedList(b,6)= newOutDomain(1,2);
                    pureMergedList(b,7)= newOutDomain(1,3);
                    pureMergedList(b,8)= newOutDomain(1,4);
                    pureMergedList(b,11)= newOutDomain(1,5);
                   outRow = b;
                  end
               end%for
             else
             % clowning previous output if not the first new one
               if(outRow ~=0 )
                 pmlsize=pmsize(1);
                 pmlsize=pmlsize+1;
                 pureMergedList(pmlsize,:)=pureMergedList(outRow,:); %outRow is the index
of previous row calculated in the first cycle
                 pureMergedList(pmlsize,5)= newOutDomain(1,1);
                 pureMergedList(pmlsize,6)= newOutDomain(1,2);
                 pureMergedList(pmlsize,7)= newOutDomain(1,3);
                 pureMergedList(pmlsize,8)= newOutDomain(1,4);
                 pureMergedList(pmlsize,11)= newOutDomain(1,5);
               end
```

```
            end
        %adding new rows for the new inputs
         end%FOR
         pureMergedList=cat(1,pureMergedList,pureMergedList1)
     end%if size domain ~= 0
    end % for(d)
     usedInputs=cat(2,usedInputs,thisLevelUsedInputs)
end % for(k)


%------------------------------------------------------------------------
% renumbering the outputIndex and also normalize the array
mergedList =
normalizeArray(pureMergedList,inputBeginIndex,inputEndIndex,mfDomainLimit);
createFuzzyRules(ouputFileName,mergedList,dataNameList);


end%for (outputs)

%first = s(:,:,1,1);
%           t = 1:size(s(1));
%           figure(1);
%           plot(t, first, 'g');
%           legend('first');
```

دانشگاه صنعتی شریف

دانشکده‌ی مهندسی کامپیوتر

پایان‌نامه‌ی کارشناسی

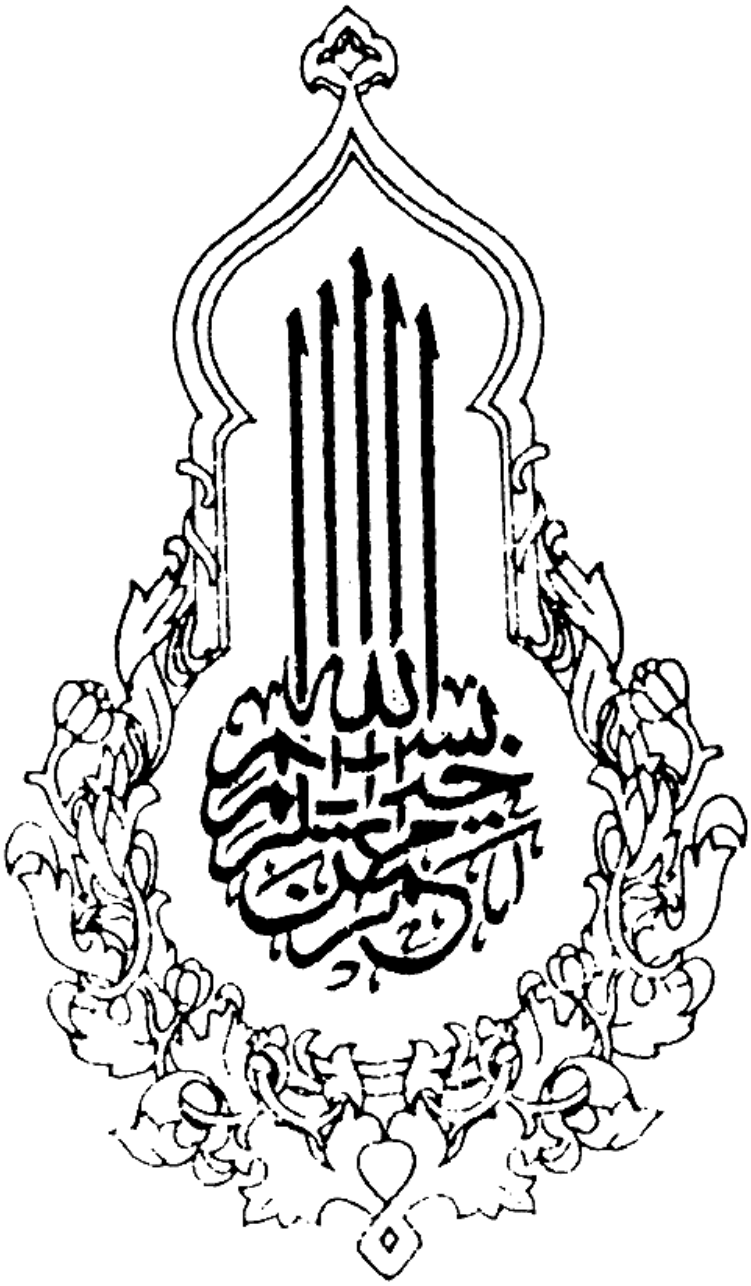
ایجاد قابلیت راه رفتن در روبات دوپا با استفاده از روش یادگیری فعال (ALM)

پوژن ضیائی (دانشجوی لیسانس)

استاد راهنما:

دکتر سعید باقری شورکی (استادیار)

شهریور ۱۳۸۴


بسم الله الرحمن الرحیم
انا فتحنا لک فتحا مبینا